Grid-Based Coverage Path Planning for Multiple UAVs in Search and Rescue Applications

by

Welri Botes

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electrical and Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr JAA Engelbrecht
Co-supervisor: Dr JC Schoeman

March 2023

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....March 2023.....

Abstract

This thesis investigates the problem of automated search and rescue (SAR) using multiple unmanned aerial vehicles (UAVs). The problem is formulated as a distributed and offline multi-robot coverage path planning (MCPP) problem. Using multiple robots generally reduces the time to detect survivors, who are assumed to remain stationary for the duration of the automated search. The survivor detection is assumed to be performed using downward-facing onboard cameras. The UAVs are assumed to fly at a constant altitude during the search, which reduces the problem to two dimensions. The environment is assumed to be known and is discretised to a grid based on the required ground sampling distance (GSD) to guarantee survivor detection. The static obstacles in the environment are represented by occupied cells. The free cells in the search area is divided into equal-sized, contiguous sub-regions to be searched by individual UAVs. This eliminates the need for explicit collision avoidance between the UAVs. A closed-loop coverage path is then generated for each subregion using a spanning tree coverage (STC) technique. The coverage paths are modified to account for the dynamic constraints of the UAVs. A central deployment strategy was developed so that all the UAVs take off and land at the same location. Flight schedules and a refuelling protocol were also developed to account for the limited endurance of the UAVs. The automated search approach using multiple UAVs was tested and evaluated in simulation using topographic maps of real-world locations representative of ground, mountainous, and marine environments. The simulation results show that the UAVs would be able to cover the search area in a reasonable amount of time, with favourable survivor detection times. The system should therefore be feasible for practical implementation.

Opsomming

Hierdie tesis ondersoek die probleem van geoutomatiseerde soek-en-redding (SAR) met behulp van veelvuldige onbemande lugvoertuie (UAV's). Die probleem is geformuleer as 'n verspreide en aanlyn multi-robot dekkingspadbeplanning (MCPP) probleem. Die gebruik van veelvuldige UAVs verminder oor die algemeen die tyd wat dit neem om oorlewendes op te spoor, met die aanvaarding dat hulle op een plek bly vir die duur van die outomatiese soektog. Die opsporing van oorlewendes word tipies uitgevoer met kameras wat afwaarts wys. Daar word aanvaar dat die UAV's tydens die soektog op 'n konstante hoogte vlieg, wat die probleem tot twee dimensies vereenvoudig. Dit word aangeneem dat die omgewing bekend is en gediskretiseer word na 'n rooster gebaseer op die vereiste grondmonstersafstand (GSD) wat nodig is om oorlewende opsporing te waarborg. Die statiese hindernisse in die omgewing word verteenwoordig deur besette selle. Die vrye selle in die soekgebied word verdeel in gelyke, aangrensende substreke om deur individuele UAV's deursoek te word. Dit elimineer die behoefte aan eksplisiete botsingvermyding tussen die UAV's. 'n Geslote-lus dekkingspad word dan vir elke substreek gegenereer deur 'n spannende boom dekking (STC) tegniek te gebruik. Die dekkingspaaie word aangepas om voorsiening te maak vir die dinamiese beperkings van die UAV's. 'n Sentrale ontplooiingstrategie is ontwikkel sodat al die UAV's op dieselfde plek opstyg en land. Vlugskedules en 'n brandstofhervullingsprotokol is ook ontwikkel om die beperkte brandstofvermoÃ« van die UAV's in ag te neem. Die outomatiese soekbenadering met behulp van veelvuldige UAV's is getoets en geÃ«valueer in simulasie met behulp van topografiese kaarte van werklike liggings wat verteenwoordigend is van grond-, berg- en mariene omgewings. Die simulasie resultate toon dat die UAV's die soekgebied binne 'n redelike tyd sal kan dek, met gunstige oorlewende opsporingstye. Die stelsel behoort dus uitvoerbaar te wees vir praktiese implementering.

Acknowledgements

I would like to express my sincere gratitude to both my supervisors, Dr JAA Engelbrecht and Dr JC Schoeman. Without their guidance and expertise, this project would not have been possible.

In particular I would like to thank Pieter Goos and Robert Waller from the Electronic Systems Laboratory for their continued advice and support. A special thanks as well to my husband and fellow engineer, Daniel Lourens.

Contents

List of Figures

List of Tables

Nomenclature

Constants

g =	$9.81\,\mathrm{m/s^2}$

Sets

$\mathcal{E}$	Search environment set
$\mathcal{O}$	Set of obstacles in the environment
$\mathcal{S}$	Search environment set excluding obstacles

Variables

x	Position in environment relative to x-axis	[m]
y	Position in environment relative to y-axis	[m]
ψ	Aircraft yaw angle	[deg]
ϕ	Aircraft roll angle	[deg]
θ	Aircraft pitch angle	[deg]
X_I	Initial state of a UAV	
X_P	Planned path of a UAV	
$X_i(t)$	Position of UAV at a specific instance in time	
X_g	Position of the target in the environment	
P_i	Path length of the i^{th} UAV	[m]
m	The number of manoeuvres that make up a path	
V_f	Constant UAV flying speed	[m/s]
$r_{\min}$	Minimum turning radius of UAV at constant velocity	[m/s]
$\phi_{\max}$	Maximum roll angle of aircraft	[deg]
ℓ	UAV Manoeuvre length	[m]
R	Largest dimension of UAV from its centre of mass	
R_e	Exclusion zone radius from UAV centre of mass	
α	Camera sensor to lens angle	[deg]
H	Height from camera lens to ground	[m]
FOV_x	Field of view (on ground) corresponding with w_{len}	[m]

FOV_y	Field of view (on ground) corresponding with h_{len}	[m]
AOV	Camera angle of view	[deg]
f	Camera focal length	[mm]
h_{len}	Camera sensor height	[mm]
w_{len}	Camera sensor width	[mm]
GSD	Ground sampling distance of camera	[cm/px]
px_w	Number of pixels along width of image for a camera	[pixels]
px_h	Number of pixels along height of image for a camera	[pixels]
H_{max}	Maximum flying height to guarantee target detection	[m]
GSD_{max}	Maximum GSD to guarantee target detection	[cm/px]
L	Dimension associated with complete coverage calculation	[m]
H_{min}	Minimum flying height to guarantee complete coverage	[m]
Δh_g	Topographic variation in environment	[m]
$h_{g_{\text{max}}}$	Altitude of highest point in topography	[m]
$h_{g_{\text{min}}}$	Altitude of lowest point in topography	[m]
H_f	Constant UAV Flying altitude	[m]
$H_{f_{\text{max}}}$	Maximum UAV Flying altitude	[m]
$H_{f_{\text{min}}}$	Minimum UAV Flying altitude	[m]
$V\text{max}$	Suggested maximum UAV flying speed	[m/s]
$V\text{stall}$	Stall speed of the UAV	[m/s]
l	Dimension of square small cell	[m]
%Overlap	Cross-track overlap	[%]
t_s	Time to execute straight line manoeuvre	[s]
t_c	Time to execute 90 degree turn	[s]
n_s	Number of straight line manoeuvres in coverage path	
n_c	Number of 90 degree turns in coverage path	
k_t	Safety factor to adjust time for energy consumption	
$\text{cells}_{\text{f}}\text{ree}$	Number of free cells	
$\text{cells}_{\text{m}}\text{ax}$	Maximum number of cells allowed per sub-region	
$(n_r)_{\text{eq}}$	Equivalent number of UAVs	
$(n_r)_{\text{avail}}$	Number of physically available UAVs	
T_p	Predicted flight time on one refuel	[s]
T_m	Time to complete manoeuvres outside of coverage path	[s]
T_f	Time to complete coverage path	[s]
$(T_f)_e$	Energy consumed to complete coverage path	[s]
T_T	Time to complete take-off manoeuvre	[s]

T_L	Time to complete landing manoeuvre	[s]
T_D	Time to complete departure manoeuvre	[s]
T_A	Time to complete approach manoeuvre	[s]
T_W	Time to complete airborne wait cycles	[s]
H_T	Take-off height	[m]
V_c	UAV climb rate	[m/s]
V_s	UAV sink rate	[m/s]
P_D	Departure path length	[m]
P_A	Approach path length	[m]
S	Square root of the number of cells in perimeter configuration	
N_t	Total number of cells in the environment	
N_o	Number of occupied cells in the environment (obstacles)	
T_{tot}	Time to complete total flight plan	[s]
$(T_{\text{tot}})_e$	Energy consumed to complete total flight plan	[s]

DARP Matrices and Variables

A	Assignment matrix
E_i	Evaluation matrix for i^{th} robot
m_i	Correction multiplier for i^{th} robot
c	Positive constant multiplier
k_i	Number of cells assigned to i^{th} robot
f	Desired number of cells per sub-region
R_i	Matrix of spatially connected cells to i^{th} robot
r	Index for cells in R_i
Q_i	Matrix of cells not spatially connected to i^{th} robot
q	Index for cells in Q_i
C_i	Connectivity matrix multiplier
Z_i	Randomized matrix multiplier
$\Delta_{\max}$	Maximum allowed discrepancy overall
Δ_{th}	Maximum allowed discrepancy in current iteration
iter	Current iteration
$\text{iter}_{\max}$	Maximum allowable iterations

Common Subscripts

x	The x dimension of the environment
y	The y dimension of the environment
k	Waypoint index

i	Robot index
j	Robot index

Acronyms

ACO ant colony optimization

AI artificial intelligence

ANFIS adaptive-network-based fuzzy interference system

CPP coverage path planning

DARP divide areas algorithm for optimal multi-robot coverage path planning

DEM digital elevation model

DGPS differential GPS

DJI Da-Jiang Innovations

DSM digital surface model

DTM digital terrain model

EMILY emergency integrated lifesaving lanyard

FOV field of view

GA genetic algorithm

GIS geographic information system

GM-VPC geodesic-manhattan voronoi-partition-based coverage

GPS global positioning system

GSD ground sampling distance

IAMSAR International Aeronautical and Maritime Search and Rescue

ICAO International Civil Aviation Organization

IMO International Maritime Organization

inSAR synthetic aperture radar interferometry

LiDAR light detection and ranging

MCPP multi-robot coverage path planning

MFC multi-robot forest coverage

MOPP multi-objective path planning

MST minimum spanning tree

MSTC multi-robot spanning tree coverage

NASA National Aeronautics and Space Administration

PO-MDP partially observable markov decision process

PRM probabilistic roadmaps

PSO particle swarm optimization

QoS quality of service

RCC rescue coordination centre

RRT rapidly exploring random trees

RSC rescue sub-centre

SAR search and rescue

SIC Simultaneous Inform and Connect

SLAM simultaneous localization and mapping

SRR search and rescue region

SRU search and rescue unit

STC spanning tree coverage

UAV unmanned aerial vehicle

UGV unmanned ground vehicle

US United States

VTOL vertical take off and landing

Chapter 1

Introduction

1.1 Background

Unmanned aerial vehicles (UAVs) have gained popularity in various applications. Initially, UAVs were just remotely operated by a ground pilot, but in recent years they have become increasingly automated. Examples of applications where UAV automation has been used include, but are not limited to, structure inspections [10], smart farming [11], disaster management [12], power line inspections [13], surveillance [14] and wildfire tracking [15]. Most of these applications used multi-rotor UAVs, particularly quad-rotors. However, the term UAV also encompasses other aircraft types, such as fixed-wing (aircraft) and rotary-wing (helicopter) UAVs. Hybrid UAVs also exist that contain both rotary-wing and fixed-wing components.

UAVs could potentially be used to support search and rescue (SAR) operations by performing aerial searches. Their ability to fly over landscapes and around three-dimensional structures gives them a considerable advantage over unmanned ground vehicles (UGVs). Their relatively high altitudes also make them well suited for automated search applications.

Perhaps the most notable example of using UAVs to perform automated search and rescue is a project by DroneSAR that uses DJI drones [1]. DroneSAR uses one drone per SAR operation. Their implementation includes a mobile application that allows a user to designate a search area manually. Figure 1.1 shows a screenshot of their mobile application. Once the search area has been designated, the drone performs a back-and-forth manoeuvre across the area to achieve coverage. The search operation can be halted if the imaging system detects a possible target in the area. The drone can be switched to manual flight mode for closer inspection and the coordinates of the target, for example a person in distress, can be sent to the SAR team. According to DroneSAR, it takes a five-person rescue team two hours on average to locate a person in one square kilometre on land, while their drone was able to perform the same task in under 20 minutes.

Building on the DroneSAR example, the next logical step would be to use multiple UAVs to cooperatively search a designated search area. This would reduce the time taken to search a given search area, or would allow a larger search area to be covered in the same time. The development of such a system would therefore be a valuable contribution to supporting search and rescue operations.

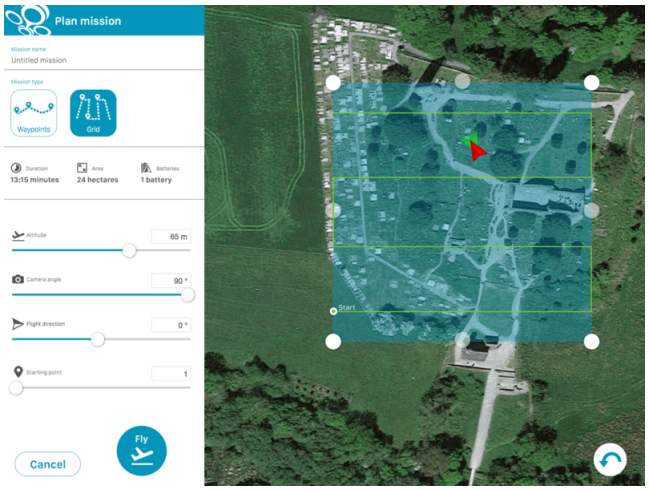


Figure 1.1: DroneSAR Mobile Application Showing Coverage Plan. [1]

1.2 Research Goal

To develop an automated search and rescue (SAR) approach with multiple UAVs using a distributed, offline coverage path planning (CPP) method.

1.3 Research Objectives

The goal of this research is to develop an automated path planning algorithm for multiple UAVs. These UAVs are required to cooperatively search an environment for survivors in a search and rescue operation. The UAV paths need to be developed to completely cover the environment in question, while avoiding collisions between UAVs as well as with the static environment.

Dynamic constraints of the UAVs need to be accounted for, as well as operational constraints. Dynamic constraints include the forward speed and associated minimum UAV turning radius. Operational constraints include the endurance limitations of the UAVs, the need to refuel to cover an environment, and the constraint of taking off and landing from a central base of operations

or landing strip. The limitations of an onboard camera in terms of coverage capacity and resolution for survivor detection should also be taken into account.

Based on the research goal, a set of research objectives are formulated:

1. Review existing literature regarding search and rescue, as well as existing solutions where UAVs are used to support search and rescue operations.
2. Conceptualise and model the search and rescue problem. Address the full search and rescue scenario and its components, namely the search area, the terrain, the UAVs and the survivor(s) within the environment.
3. Develop an automated search and rescue algorithm that plans coverage paths for multiple UAVs. The UAVs must search the designated area while adhering to their dynamic constraints, which is dependant on the forward speed of the UAV. General operational constraints should be considered, including the endurance constraints of the vehicle as well as the central deployment constraint. Lastly, the limitations of the onboard cameras for survivor detection should be accounted for.
4. Develop a UAV take-off and landing procedure as well as flight schedules for the UAVs with the assumption of a central landing strip or base of operations.
5. Build a platform to simulate possible search and rescue scenarios. These simulations should be used to test the algorithm as a means for automating search and rescue with multiple UAVs.
6. Evaluate the performance of the algorithm in simulated, randomly generated environments as well as mapped, real-world environments. Assess the feasibility of using the algorithm in real-world search and rescue operations.

1.4 Proposed Solution and Contributions

The automated SAR problem is formulated as a coverage motion planning problem. It is solved using an offline, grid-based and distributed coverage path planning (CPP) approach for multiple UAVs.

The demarcated search area is discretised into a grid with static obstacles which are represented as occupied cells. The free cells in the grid representation are divided into equal-sized, contiguous sub-regions to be searched by individual UAVs. The algorithm used for this is called the divide areas algorithm for optimal multi-robot coverage path planning (DARP).

The resulting sub-regions are roughly equal in size. This means that the time to cover each sub-region is similar as well. Covering the individual sub-regions can also be done independently. Collision avoidance is not necessary,

since each UAV can search its assigned region without traversing the region of another UAV.

The initial positions of the UAVs, where they begin their search, are chosen to be along the perimeter of a central base of operations. The sub-regions are formulated so that one UAV begins its search within each region.

A closed-loop coverage path is generated for each sub-region using a spanning tree coverage (STC) technique. Each sub-region can then be completely covered by one UAV. Dynamic constraints of the UAV are then taken into account by introducing a minimum UAV turning radius based on the constant forward speed of the UAV. The original paths are modified to incorporate this turning radius.

The sub-region size is limited by the physical UAV endurance, so there may be more sub-regions than UAVs. When this is true, multiple regions are assigned to one actual UAV for searching, but the UAV refuels in between respective sub-region searches. This is made possible by using a central ground station for take-off, landing and refuelling. The initial robot positions, of which multiple may be associated with one actual UAV, are arranged in a perimeter configuration around the ground station.

The UAVs take off sequentially from the base of operations and fly to their initial positions around the ground station perimeter. Once they are each within a respective sub-region, they follow their closed-loop paths, eventually reaching the initial positions again, from which they land back to the base of operations. If more sub-regions are assigned to these UAVs, they would be refuelled or recharged and start the process again for a different set of unsearched sub-regions.

Figure 1.2 shows the result of using the DARP algorithm for a hypothetical environment. The central ground station is shown as an obstacle with the robot initial positions on its perimeter. They are shown as black dots with white at their centre.

There are only two available UAVs and the sub-regions that are allocated to them are represented in blue and orange respectively. Multiple regions, with multiple initial positions, are assigned to each UAV. They can be distinguished by the different shadings in each colour. The target location is shown as an "X." Note that this is not a known value to the UAVs in a real-world scenario.

The UAV will execute a series of closed loop coverage paths in each of its regions as shown in Figure 1.3, starting in the most darkly shaded region. This figure shows a snapshot of the environment at the moment of survivor detection. In this example, survivor detection occurred in the final orange-sub-region that was searched. The paths to land and take-off at the ground station are not shown here, but are implied.

In summary, the the multiple robot, automated SAR problem is formulated as an offline CPP problem. This is solved using a grid-based, two-dimensional, distributed approach that results in approximately complete coverage of a search area.

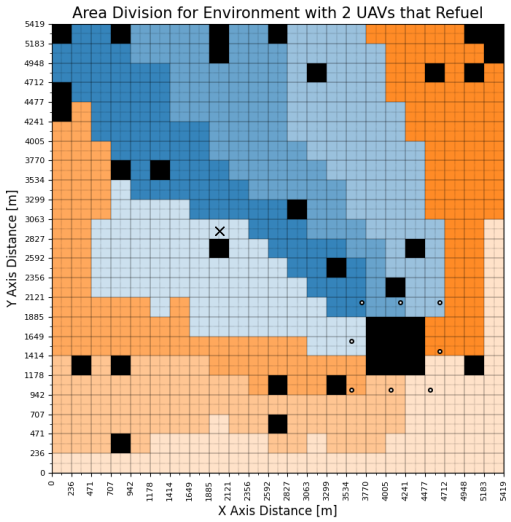


Figure 1.2: Environment map showing the area division by the DARP algorithm in an example environment where there are two UAVs that refuel three times.

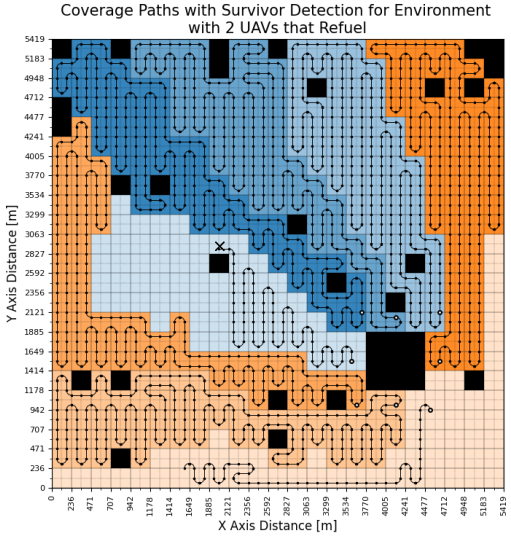


Figure 1.3: Environment map showing the coverage paths of two UAVs in an example environment up until survivor detection occurs after the final refuel.

Some contributions include:

1. The multi-robot, coverage path planning approach developed by Kapoutsis et al. [6] was applied to the SAR problem and was expanded to take the operational constraints and dynamic constraints of the UAVs into

account.

2. A procedure was developed for choosing the appropriate grid size when discretising the search area based on the specifications of the UAVs and their onboard camera payloads. The relevant design equations for doing so were also formulated.

3. A procedure was developed to generate paths for the departure and approach of the UAVs. These entail the travel between the take-off and landing points at the centre of the base of operations, and the initial positions of the sub-region coverage paths around the perimeter of the base of operations.

4. Plausible flight schedules and refuelling protocols were developed for the deployment of the UAVs from a central base of operations.

5. The automated search and rescue approach was tested and evaluated in simulation with search areas based on real-world maps. The dynamic and operational constraints of real-world UAVs and their onboard camera payloads were accounted for.

1.5 Scope of Project

The multiple UAV automated search approach proposed in this thesis is intended to function as part of a SAR operation. It may be part of a much larger search plan involving various specialized teams.

In any specific SAR scenario, the search coordination team would need to decide how a team of UAVs would be the most useful. They may be used to search a sub-region of a much larger search area. They may also be used as aerial assistance to a manual search happening in tandem. The focus of this research project is the development of an algorithm to generate paths for the UAVs to follow in their automated search.

The automated SAR approach developed in this project was tested and evaluated in simulation with search areas based on real-world maps, and with dynamic and operational constraints representative of real-world UAVs and onboard camera payloads. However, the project did not go as far as implementing the system on real flight computers and performing practical flight tests with the physical UAVs. This could be the subject of a future research project.

The development of the onboard imaging systems and the target detection algorithm are considered to be outside the scope of this project. The target detection is abstracted by assuming that the target is detected when it is in range of the UAV. The algorithms are assumed to favour a false positive over a failed detection.

The development of the flight control systems for the UAVs are also considered to be outside the scope of this project. It is assumed that the UAVs are equipped with flight control systems that allow them to follow given reference trajectories.

The multi-robot coverage path planning (MCPP) approach implicitly provides obstacle avoidance for static terrain features in the search area and collision avoidance between the UAVs performing the cooperative search.

Collision avoidance for dynamic obstacles is not accounted for. It is assumed that the UAVs are equipped with short-term collision avoidance systems, similar to the one presented by Meiring et al. [16]. The development of this short-term collision avoidance system is considered to be outside the scope of this project.

A flight schedule is developed so that the UAVs can take-off and land sequentially and wait in holding patterns if necessary. This prevents UAV collisions within the take-off and landing zones. This zone is also assumed to be clear of static obstacles. Collision avoidance in the event that there are static obstacles is beyond the scope of the project.

1.6 Limitations

This proposed solution in this thesis does not accommodate contexts like cave, urban or combat SAR. An aerial search would likely also be inappropriate for a region with poor visibility, such as a dense forest or a region with bad weather. Regions like these are assumed to be excluded from the aerial search by the search coordination team.

The approach used in this thesis discretises the search area into a grid-based, two-dimensional environment. This facilitates the area division and the path generation, but also introduces some limitations.

To generate a two-dimensional grid with a constant grid cell size, a constant altitude search must be assumed. For a search area with a large variation in terrain altitude, it would be advantageous to be able to perform searches at different altitudes.

The grid-based approach also makes complete coverage of the actual continuous environment more challenging. Obstacles must either be over- or underestimated to mark grid cells as free or occupied, which means that the grid-based environment is only an approximation of the actual continuous environment. Although complete coverage of the grid environment can be achieved, complete coverage of the the actual continuous environment can only be approximately achieved [17].

The proposed coverage path planning approach assumes a static target, or at least a target that is moving relatively slowly compared to the UAVs. Target detection can be guaranteed if the target is static, but cannot be guaranteed for a moving target. The assumption of a static or slow-moving target is

reasonable, since individuals in need of search and rescue are encouraged to stay in one location, or would likely be on foot.

The proposed approach assumes that the search area is a known environment and has been mapped prior to the search. This offline coverage path planning approach only considers static obstacles, and does not consider dynamic obstacles such as other aircraft that are operating in the same search area. The approach assumes that any dynamic obstacles would be handled online by a short-term collision avoidance system.

The dynamic constraints of the UAV, namely the forward speed and minimum turning radius, impose a required minimum flying height to produce complete coverage. On the other hand, the resolution of the onboard camera imposes a maximum flying height constraint above which target detection is not ensured. The UAV and camera combination must therefore be carefully selected to produce a feasible range of flying heights.

To ensure that a grid is completely covered while the UAV is turning, a grid cell size that is smaller than the onboard camera's field of view must be used. This introduces a general overlap in the coverage. The disadvantage is that there is some redundant coverage. The advantage is that the overlap increases the chance of detecting a target that is not stationary, but moving slowly between grid cells. Furthermore, the overlap could address the approximate coverage, making it tend towards complete coverage.

A set of manoeuvres is needed to get UAVs from the ground station to their initial positions. These are arranged in a perimeter configuration around the ground station, which is viewed as an obstacle. Currently the approach is limited to eight UAVs, but it can be expanded for more robots if required. This is a limiting configuration though and there may be other configurations better suited to a scenario.

The proposed solution assumes that all UAVs are deployed from a central base of operations. However, other deployments strategies are also possible, for example where the UAVs are deployed from multiple ground stations, and perhaps do not have to land at the same ground station from which they took off. The proposed solution would have to be adapted for the specific deployment strategy. This could be the subject of future research.

1.7 Thesis Outline

A brief overview of the structure of this thesis, beyond the introduction, is given below:

- **Chapter 2 - Literature Review.** This chapter establishes some context and background information on search and rescue operations, provides an overview of how robots and UAVs have been used for search and rescue in the past, and performs a literature review on motion planning, single robot CPP, and multiple robot CPP.

- **Chapter 3 - Conceptualization and Modelling.** This chapter conceptualises and models the search and rescue scenario and its major elements, namely the search area, the terrain, the UAVs, and the target. How target detection and collisions are modelled is also addressed.

- **Chapter 4 - System Overview.** This chapter provides an overview of the proposed solution as well as a brief description of each of its components and their relationships.

- **Chapter 5 - Environment Representation.** This chapter deals with the first phase of the proposed solution. It details the process of selecting a type of UAV and camera for an environment, as well as choosing an appropriate flying speed and height. It goes on to show the process used to discretise a specific environment into a grid.

- **Chapter 6 - Divide Areas Algorithm.** This chapter describes the method for dividing the designated search area into equal sub-regions using the DARP algorithm.

- **Chapter 7 - Sub-region Coverage Technique.** This chapter describes the method used to generate the path for an individual UAV in its allocated sub-region using a spanning tree coverage (STC) algorithm, and then modifying the path to account for the dynamic constraints of the UAV.

- **Chapter 8 - Central Deployment and Scheduling.** This chapter considers the practical implications of deploying the UAVs from a central base of operations. Procedures are developed to generate the take-off and landing paths between the central ground station and the search path starting locations at its perimeter. Flight schedules and refuelling protocols are developed to allow the UAVs to take off and land sequentially.

- **Chapter 9 - Monte Carlo Simulations.** This chapter describes the monte carlo simulations that were performed to test and evaluate the proposed automated search approach. The simulation results are presented and discussed.

- **Chapter 10 - Conclusions and Future Work.** Finally, the thesis concludes with a summary of the work done and how the research objectives were met. Recommendations for future research and improvements are also provided.

Chapter 2

Literature Review

A literature review was performed to obtain background information on the search and rescue (SAR) problem, to get an overview of how robots and UAVs have been used for search and rescue in practice and in previous research, and to review existing motion planning and coverage path planning (CPP) techniques that could be used to perform automated searches using multiple UAVs. CPP is divided into single robot coverage path planning and multi-robot coverage path planning (MCPP). MCPP is further divided into three categories: distributed offline MCPP, non-distributed offline MCPP, and online MCPP. At the end of the chapter, the key conclusions from the literature review are summarised, and are used to inform the research decisions for this project.

Section 2.1 provides background information on the SAR problem, including SAR organizations, the stages of SAR, and different types of SAR. Section 2.2 provides an overview of how robots and UAVs have been used for SAR in practice and in previous research. Sections 2.3 and 2.4 provide some background theory on general motion planning and CPP. Section 2.5 reviews techniques for single robot CPP techniques. Section 2.6 reviews techniques for MCPP. Section 2.7 summarises the key conclusions from the literature and the research decisions that were made.

2.1 Search and Rescue

This section discusses the principles and conventions surrounding SAR, providing context for the general SAR problem. Section 2.1.1 discusses SAR operations in the global sense, with mention of the main governing bodies and documentation. The different stages of a SAR operation are discussed in Section 2.1.2 and the different types of SAR, relating specifically to the type of environment, are discussed in Section 2.1.3.

2.1.1 Search and Rescue Globally

Search and Rescue operations have historically been performed using teams of people on the ground doing some type of planned search. As advancements came in technology, it began being used to assist. Technologies such as manned boats and aircraft are often used to offer assistance in SAR operations. Other less common technologies, like thermal cameras, can also be utilised to speed up a search.

The information in this section is taken from three sources. Volume I and Volume II of the International Aeronautical and Maritime Search and Rescue (IAMSAR) manual are used along with a set of slides set up by the International Civil Aviation Organization (ICAO) regarding the best practices for search and rescue. [18, 19, 20]

SAR often involves maritime and/or aeronautical components, either as a vessel in distress or as a search and rescue unit (SRU) in a SAR operation. For this reason, the International Maritime Organization (IMO) and International Civil Aviation Organization (ICAO) have become the two main authorities on SAR worldwide.

Both organizations envision a global SAR network, where the globe is divided into search and rescue regions (SRRs), each with an associated rescue coordination centre (RCC) responsible for any SAR operation within that region. The overarching goal is that regardless of where a person is in distress, SAR services will be available. Any country or state that has agreed to this global vision is expected to follow certain procedures and protocols during a SAR operation. They are also expected to adhere to a certain organizational structure and should always have specific equipment available.

There is a manual, that outlines all these requirements and protocols, called the IAMSAR manual. This manual is approved by the ICAO and IMO and aids in coordinating operations that have both aeronautical and maritime components. This manual consists of three volumes, each aimed at a different component of the SAR system. Volume I, the Organization and Management volume, is aimed at SAR system managers and addresses SAR as a global concept. In particular it addresses the responsibilities of RCCs and cooperation between neighbouring regions. Volume II, which is the Mission Coordination volume, gives the guidelines for coordinating multiple organizations and regions for a SAR operation. This is aimed at rescue coordination centre (RCC) and rescue sub-centre (RSC) personnel. Volume III is a direct guide for SRUs and the protocols they should follow, as well as what vessels should do if they themselves are in distress.

In the global SAR system perspective, the globe is divided into several SRRs. A national or regional agency, an RCC, is assigned in each region and is responsible for SAR services within this region. This service is expected to be provided promptly and effectively, and without regard for circumstance or nationality. States are expected to share resources and facilities in such a way

that the SAR operation is conducted in a coordinated and efficient manner.

The maritime and aeronautical SRRs can be slightly different, but are generally quite similar. The benefit of allocating these regions is that distress signals can be automatically routed to the relevant RCC to allow for a swift emergency response.

2.1.2 The Stages of Search and Rescue

The stages of a SAR operation are detailed in Volume II of the IAMSAR manual [19]. Receiving a distress call is generally the first step when it comes to a SAR operation. Being reachable and having open lines of communication is one of the most important technological components of a SAR operation. Distress alerts can be relayed to the relevant RCC via coastal radio stations, air traffic service units, land earth stations, other RCCs, and numerous other resources available.

Communication during a SAR response is also key to ensure all parties involved stay informed. Portable radios, mobile devices and satellite phones can be vital to an effective SAR operation. SAR aircraft and vessels also communicate on specific frequencies to ensure no interference.

Depending on the equipment, there may also be homing capabilities. Many vessels and aircraft have some method of alerting others of their location. Civil aircraft usually have an emergency locator transmitter, for example.

Homing in on the location of survivors is crucial in an emergency situation. Extreme conditions or injuries are not unusual in these scenarios and it is imperative that a person be located as quickly as possible so that medical services can be provided if necessary and further injury, or possibly death, can be avoided.

The IAMSAR manual divides a SAR response into five stages. The first stage is referred to as the awareness stage. This is where a SAR organization becomes aware of people in distress via its communication channels.

Once a possible SAR situation has been identified, the next stage involves identifying the appropriate emergency phase. This is called the initial action stage. The emergency phase can have one of three classifications: uncertain, alert or distress phase. This classification can change as new information about a situation becomes available. The type and extent of the SAR response necessary can then be ascertained by evaluating the urgency of the situation.

Proper planning and coordination of a SAR response is crucial towards its success. Therefore the next stage, after initial action, is the planning stage. A number of critical decisions need to be made at this stage, like which area needs to be searched and what resources, equipment and facilities need to be utilised.

Generally the planning stage begins with determining the probable survivor locations and using this information to demarcate an area that needs to be searched. Information that is often used in this estimation is: the results

of previous searches in the area, environmental conditions like ocean currents or wind patterns, the last known location of survivors, the location of the distress incident, the original travel route of the survivors, the state of the survivors and whatever craft they may have been travelling in, and possible environmental hazards in the area.

The distribution of facilities and SRUs in the area is then used to generate a search plan. Generally, search parties do not opt for a systematic search, particularly when there is a large area to cover. Instead they have a probabilistic approach, searching areas where the survivors are more likely to be first.

This method is favoured in partial because it has been found that after three days, the likelihood of survival of a person in distress rapidly decreases. If a person is injured, likelihood of survival decreases by 80% in the first 24 hours. Finding survivors swiftly is therefore crucial in SAR.

Once a plan has been formulated, it can be put into action in the operations stage. Here the practical search operation is executed. In this stage it is intended that the survivors are found, provided with any immediate assistance they may require, and returned to safety. This is followed by the final stage, called the conclusion stage. This is when survivors are no longer in distress.

The conclusion stage may be reached prematurely if it is found that there is no distress situation, or that the search is no longer necessary because the likelihood of survival of those in distress is practically zero. A search may also be abandoned if it is found to be too dangerous for the search crew. Regardless of the reason, a search operation is always eventually terminated, and the entire search process then needs to be documented, in detail, by the RCC.

2.1.3 The Types of Search and Rescue

The nature of a search and rescue operation has a big impact on the procedures followed and facilities used during a search. The terrain is a big part of this. Volume II of the IAMSAR manual discusses the effects of the terrain on a search operation [19].

A mountainous region, for example, tends to have thin air and turbulence for aircraft. Therefore a helicopter search is generally unsuitable. Fixed wing aircraft at higher altitudes are most likely better suited in this situation, but then a target cannot be hoisted to safety and would have to be extracted with some other skilled team. Similar problems may occur in extreme weather conditions.

The more complicated the terrain, the more experienced and skilled a search team needs to be. Many types of services may need to be utilised, some of which include volunteers, forest services, mountain rescue teams, para-rescue teams, law-enforcement, firefighters and even skiing clubs.

In general, there is always some danger to the people in aircraft or on other search vessels, as well as to a search team on the ground. Evaluating the dangers in the area are crucial to providing effective aid to those in distress, whilst avoiding putting a search team in unnecessary danger.

It is possible to divide SAR operations into certain categories according to the type of environment in which the search needs to take place. According to an article on a Search and Recovery Engineering website [21], a common practice is to divide SAR into the following categories: ground rescue, mountain rescue, maritime rescue, urban rescue and combat rescue.

Ground rescue, which is typically carried out by volunteers or local law enforcement, concerns people in distress on land. This kind of search could involve people who have run away from home, people who are lost, people who are in distress due to weather conditions or for any other number of reasons. It can also occur in both rural and built-up areas.

Mountain rescue often involves people in distress in caves or mountainous terrain. This may be as a result of a caving or rock climbing accident. Specialised teams with mountain climbing equipment may be required to assist those in distress. This is often dangerous work, since the terrain can be treacherous and cave networks are often not mapped in detail.

Maritime rescue most often involves marine vessels in distress. The search is generally conducted by institutions like the navy or coast guard. Marine vessels and aircraft, for example helicopters, are used to locate and retrieve those in distress.

Urban rescue, which should not be confused with ground search and rescue, involves locating people in distress in urban regions where a disaster has occurred. This often involves building collapses due to natural disasters or other disturbances. People get trapped in rubble and extraction can often be incredibly challenging. Firefighters, medical personnel, local law enforcement and various other teams may be involved in urban rescue.

Combat rescue involves rescue in or near active war zones. This is a very specialized type of rescue operation and often involves injured combatants.

Regardless of the type of rescue operation, planning and coordination is crucial to the success of a search operation. Identifying the type of terrain, the hazards and the rescue services required is important to ensure the response is fast and those in distress are helped as soon as possible.

2.2 Robotics in Search and Rescue

Robotics for SAR is discussed in two main categories. Section 2.2.1 briefly looks at remotely operated robots and UAVs that have been used for SAR. The various roles robotics can play in SAR operations are discussed in particular. Section 2.2.2 in contrast looks at automated UAVs that help search operation with the use of path planning and optimization algorithms.

2.2.1 Remotely Operated Robots and UAVs for Search and Rescue

According to the Springer Handbook of Robotics [22], robots can be applied for SAR in a number of ways. Unless specified otherwise, the examples in this section are sourced from this handbook.

Robots can be applied in the actual search task or for mapping an area that needs to be searched. Robots can be used to remove rubble or inspect structures to determine if they are safe to traverse. They can be used for medical assistance as well, such as delivering medical supplies or helping medical officers communicate with survivors. Furthermore, they can also be used to provide logistical support such as aiding in transport of equipment and supplies.

Whatever their role, SAR robots are intended to speed up a SAR operation and assist survivors as swiftly as possible. One of the first prominent examples of where robots were used in SAR was during the World Trade Centre disaster in 2001. Unmanned ground vehicles (UGVs) were used to search for survivors in the rubble. They successfully uncovered several sets of remains and inspected the damage to the foundations.

During several large hurricanes in the United States, unmanned aerial and ground robots were used to assist in search and rescue. During Hurricane Katrina, a battery-powered fixed-wing UAV was used, as well as a battery-powered helicopter that was adapted for operating in high winds. They were used to explore difficult to reach regions, such as those cut off by debris and flooding. To identify areas that still require assistance, a Silver Fox was also used. This is an internal combustion engine, fixed wing UAV often used by the United States Navy. All the UAVs flew below regulated airspace and provided information directly to SAR responders as they surveyed various areas.

In Hurricane Rita and Hurricane Wilma, fixed-wing, internal combustion engine UAVs were also used to survey the disaster region, but they were flown in regulated airspace. These were military grade Predator UAVs that are built for endurance, but require more people to operate. They are also much larger and generally require larger landing and take-off zones.

With maritime SAR, remotely operated surface or underwater vehicles can be used to assist. One example of this is called the emergency integrated lifesaving lanyard (EMILY) [23]. This is a remotely operated robot that moves on the surface of water and acts as a buoy. It has been used successfully to help refugees in the Mediterranean and is being deployed all over the world. Researchers seek to add more autonomous capability to this robot in future.

Unmanned ground, aerial, surface-water and underwater vehicles are clearly sought after to access difficult to reach areas in SAR operations. They reduce the risk to search personnel by acting in their stead or gathering valuable information to ensure their safety and the safety of survivors.

2.2.2 Automated UAVs for Search and Rescue

This section discusses the application of UAVs for SAR. The automation component is generally in the form of path planning for the UAVs or target detection using thermal and/or visual cameras onboard the UAVs. A general discussion of the algorithms used, and the level of success achieved, is given in each case. Section 2.2.2.1 describes a technique that is being used in practice for ground and maritime SAR. The subsequent sections are application that have been tested in simulation or in practice, but are not actively being used for SAR operations.

2.2.2.1 Complete Implementation by DroneSAR

DroneSAR is an Irish company that developed a system to assist search and rescue with a DJI quad-rotor UAV [1]. They created a user-friendly application, where one can demarcate an area to be searched and it will plan its own coverage path of the area. The algorithm uses simple back-and-forth manoeuvres or a manual user input of way-points to plot a course for the quad-rotor. On-board video footage (visual or thermal) is sent to the ground station in real time. This allows the team on the ground to react quickly when a target is found.

The goal was to find survivors faster, and reduce risk to SAR teams for ground-based and maritime SAR. Based on tests with search and rescue teams, it was found that the time taken to find a victim in one square kilometre, with five people searching, is roughly two hours. Their system was then found to locate the target in under 20 minutes.

According to an informative video released by the company [24], it is currently the task of the the pilot to review footage, as the UAV is flying, and locate a target. The UAV then sends GPS coordinates to the SAR team so they can assist and retrieve the survivor(s). In future, they do intend to add automatic human detection algorithms using artificial intelligence (AI).

2.2.2.2 Artificial Intelligence Based Approaches with Multiple UAVs

San Juan et al. [25] proposed several methods to perform intelligent UAV map generation and discrete path planning for search and rescue operations. A map of the search area to be covered is divided into a grid of cells and each cell is assigned a risk/occupancy value that represents the probability for the cell to be occupied, and the potential hazard to the life of the occupant. The risk/occupancy grid indicates which cells should be visited sooner and is used to for the discrete path planning.

Four discrete path planning approaches were used to generate the way-points for an individual UAV to follow: a potential field approach, a fuzzy logic approach, an adaptive-network-based fuzzy interference system (ANFIS)

approach, and a particle swarm approach. The approach was extended to use multiple UAVs by performing the discrete path planning for two different swarm formations: free and distributed. For the free swarm formation, the discrete path planning algorithm is executed for two or more UAVs at the same time in parallel. Each UAV follows an independent path along the coverage area. When calculating the waypoints, the information about the cells that have already been visited by UAVs are shared, so that the UAVs do not plan to visit cells that have already been visited. For the distributed swarm formation, the map is divided into a number of sub-regions equal to the number of UAVs, and each UAV is assigned an area which it must cover.

The methods were tested in simulation and the results showed that the adaptive-network-based fuzzy interference system (ANFIS) approach performed the best in general, and that the distributed formation worked better than the free formation.

2.2.2.3 Online Approach with Multiple UAVs and Changing Altitudes

Sawarte et al. published several works regarding the use of UAVs for SAR in the years 2009 and 2010. Their first work concerned coordinated search operations with a swarm of UAVs [26]. They presented an online approach that uses quad-rotors to search for a single stationary target in a two-dimensional search area. The UAVs use downward facing cameras for target detection and onboard GPS for localisation. The search area is divided into a grid and each cell is assigned a value that represents the probability of the target being within the cell, creating a probabilistic occupancy grid. Each UAV maintains its own copy of the occupancy grid and updates its cell values as it explores the area. The UAVs communicate their occupancy grids with one another when they are within communication range. The UAVs search the area in a decoupled manner and apply the steepest gradient method to their occupancy grid to decide on the next cell to visit. The approach was tested in simulation and the results show that using multiple UAVs, that share information, significantly decreases the time to find the target.

In a second paper [27], the authors built on their work by including the ability to fuse multiple observations for the same cell, and by accounting for the UAVs changing altitudes.

In a third paper [28], they presented the target detection algorithm that detects the target using the video feed from the UAV's downward-facing camera. They found that the sampling rate should be chosen based on the requirements of the application. For search and rescue, the sampling rate should be chosen to minimise missed detections.

In their fourth paper [29], they investigated three different strategies for application in SAR: namely greedy heuristics, potential-based algorithms and the partially observable markov decision process (PO-MDP). These online ap-

proaches were designed to deal with information sharing limitations, collision avoidance and uncertainties in the sensor data. The approaches were tested in simulation and the results showed that PO-MDP achieved the fastest target detection.

2.2.2.4 Online Approach with One UAV and Human Detection Algorithm

Rudol and Doherty [30] presented a human body detection and geolocalisation technique for UAV search and rescue missions using color and thermal imagery. Their techniques used a single, unmanned helicopter equipped with onboard visual and thermal cameras. The unmanned helicopter executes back-and-forth motions over a search area and collects video footage. The footage is then analysed using an algorithm for human body detection. The lower-resolution thermal images are used to find the potential locations of humans, and then the higher-resolution visual images are analysed to confirm. The path planner, to formulate the UAV's back-and-forth motions, uses a motion planning framework previously developed by Wzorek et al. [31].

Wrozek et al. developed a motion planning framework for a single rotary-wing UAV which integrates two sample-based motion planning techniques, probabilistic roadmaps (PRM) and rapidly exploring random trees (RRT) together with a path following controller that is used during path execution. They incorporated an online planning method to change the UAV's flight path on the fly in response to certain dynamic changes in the environment. The dynamic changes were handled by using no-fly zones or pop-up zones which could be added or removed by a ground operator. Their system was verified through simulation and in actual flight. Their algorithm was found to detect humans at a rate of 25 Hz, and was designed to favour a false positive detection over a failed detection.

It should be noted that the motion planning framework developed by Wrozek et al. is a point-to-point path planner. Rudol and Doherty extended it to CPP by using the point-to-point path planner to perform back-and-forth motions to cover a search area.

2.2.2.5 Multiple UAV Approach with Quality of Service Requirements

Hayat et al. [32] presented a work where multiple UAVs are used for automated SAR. The problem was formulated as an MCPP problem, but with communication as an additional mission goal. A previously developed multi-objective path planning (MOPP) algorithm, called the Simultaneous Inform and Connect (SIC) algorithm [33], was used since it can be tuned to favour connectivity, coverage or both in varying degrees.

The SAR problem is broken down into three tasks: namely search, inform and monitor tasks. The search task concerns the path planner used to achieve coverage and detect a stationary target. Once the target is detected, its location needs to be transmitted to a ground station as part of the inform task. Lastly, a good quality of service (QoS) connection link needs to be set up for the monitor task so that the target location can be monitored in real time. A genetic algorithm (GA) was used to optimize the time to complete all three tasks. Replanning is used to reconfigure the UAVs once the target is detected and establish a stable link to the ground station.

Two different SIC strategies were tested in simulation. One optimizes all three tasks simultaneously, while the other optimizes the search and inform tasks first, followed by the monitor task. Using simulations, it was found that the joint optimization technique yields better results. Results also showed that favouring connectivity gives better results for a small group of UAVs and favouring coverage gives better results for a larger group. Overall, the algorithms were also shown to have faster mission completion times than similar algorithms that use connectivity as a constraint instead of a goal.

2.3 Motion Planning

This section provides an overview of general motion planning concepts and techniques found in the literature, in particular from a book on motion planning by Steven Lavalle [34].

In robotics, motion planning is the problem of converting high-level specifications of robot tasks into low-level descriptions of how the robot must move. Lavalle distinguishes between motion planning and trajectory planning. Motion planning focuses on the series of translations and rotations required to move a robot from one configuration to another within some environment, and usually ignores the dynamics and other differential constraints. Trajectory planning usually refers to the problem of taking the solution from a motion planning algorithm and determining how to execute it in a way that obeys the dynamics and differential constraints.

A motion planning problem typically consists of an robot, an environment, and a plan. A planning algorithm is used to plan the path for the robot to execute in the environment, and the robot then executes the plan. The plan can be executed in simulation or in the real world.

Motion planning problems can be classified as either continuous or discrete. Continuous planning models the robot as having continuous inputs which are used to move it through a continuous state space, and a solution is constructed by determining the appropriate input signals versus time. Discrete planning models the robot as having a finite set of actions that can be applied to a discrete set of states, and a solution is constructed by determining the appropriate sequence of actions.

Discrete planning techniques typically use search methods, such as Dijkstra's algorithm and A*, to find the optimal sequence of states and actions. Continuous planning techniques are classified into two major categories, namely combinatorial methods and sampling-based methods.

Combinatorial methods formulate continuous paths by building a discrete representation of the environment, including the robot and the obstacles within the environment. These methods are also referred to as exact since they exactly represent the original continuous problem. Combinatorial methods are complete, which means that they are guaranteed to find a solution when it exists, or correctly report failure if one does not exist. Combinatorial methods use methods like trapezoidal decomposition and Voronoi diagrams to generate discrete roadmaps, and then traverse the roadmaps using discrete search methods such as A*.

Sampling-based methods use collision detection methods to sample the continuous state space and then perform discrete searches. Sampling-based methods are resolution complete or probabilistically complete, which are weaker forms of completeness. Resolution completeness means that if a solution exists, the algorithm will find it in finite time; however, if a solution does not exist, the algorithm may run forever. Probabilistic completeness means that as the number of sampled points tends to infinity, and the probability of finding the solution tends to one. Examples of sampling-based methods include rapidly exploring random treess (RRTs) and probabilistic roadmapss (PRMs), which are single-query and multi-query methods, respectively.

Motion planning problems are also classified according to the nature of the high-level task they perform. The task could involve a single robot or multiple robots. (In the field of artificial intelligence, robots are also called agents or decision makers.) Planning with multiple robots can be quite challenging because the robots must not only avoid collisions with obstacles in the environment, but also with one another.

Figure 2.1 provides an overview of the different types of path planning according to the high-level task.

General motion planning is divided into point-to-point path planning and coverage path planning (CPP). In point-to-point path planning, the task is to move from one point to another and/or to change orientation. With CPP, the task is to cover every point in an environment. Point-to-point path planning involving multiple robots is divided into the rendezvous task and the allocation task. If the goal location is the same for all robots, then it is called a rendezvous task. If the robots have different goal locations, then it is called an allocation task.

Finally, motion planning algorithms can be classified as either offline or online. With offline planning, the path planning is performed and completed before the path execution begins. With online planning, the path planning and path execution are performed in tandem. For offline planning, complete knowledge of the environment is assumed. For online planning, the environ-

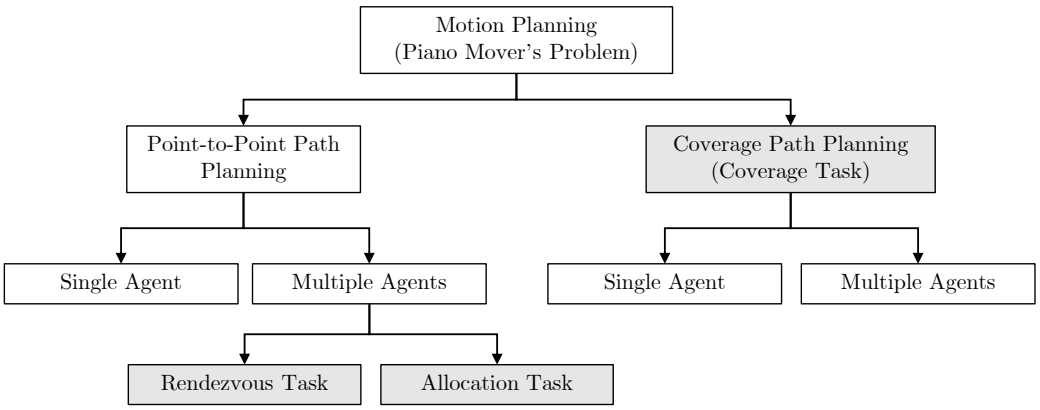


Figure 2.1: Flow diagram showing a breakdown of the different kinds of path planning as part of motion planning.

ment is sensed as the robot moves, and the plan is generated and updated while it is being executed by the robot. [35]

2.4 Coverage Path Planning

Coverage path planning (CPP) is a subset of the general motion planning problem. The coverage task refers to visiting all points within an environment as opposed to the usual start-goal type task [36]. CPP can fall into the same categories as motion planning. It can be classified as discrete or continuous, online or offline, and as a single or multiple agent problem.

CPP can be used for a number of different applications. Some past examples include its use with vacuum cleaning robots, spray painting robots [37], window cleaning robots [38], and automated lawn mowers [39]. For underwater vehicles, it can be used for the inspection of difficult-to-reach underwater structures [40] and with ground vehicles it can been used to automate field machines for smart farming [41].

A number of surveys have been done to give an overview of the literature available and progress made in the field of CPP. In 2001, Choset [17] performed a survey wherein they divide CPP into four categories: heuristic, approximate, semi-approximate and exact cellular decompositions. In later papers this is known as Choset's taxonomy, and it is widely used to categorize different types of CPP algorithms. The cellular decomposition approaches all rely on simplifying the environment to achieve provably complete coverage. Choset also briefly covers multi-robot coverage path planning (MCPP) algorithms.

Heuristics approaches use a set of rules to produce simple behaviours, such as following a wall, to cover a search area. The heuristics may work well, but do not provide any provable guarantees that ensure successful coverage.

Approximate cellular decomposition approaches use a fine grid to represent the free space to be searched. Semi-approximate cellular decomposition approaches rely on a partial discretisation of the search space where cells are fixed in width but the top and bottom (or ceiling and floor) may have any shape. Exact cellular decomposition approaches use a set of non-intersecting regions, each called a cell, whose union fills the target environment. The robot then covers each cell using simple back-and-forth motions.

In 2013, Galceran and Carreras [42] presented an updated survey of coverage path planning for robotics that reflected on the advances since Choset's survey. The survey reviewed the most successful coverage path planning methods and discussed their reported field applications. The survey also covered CPP in three-dimensional scenarios, and briefly looked at CPP where simultaneous localization and mapping (SLAM) is applied to handle localization uncertainties.

In 2019, Cabreira et al. [43] published a survey on coverage path planning with UAVs. They considered simple geometric flight patterns and more complex grid-based solutions which consider full and partial information about the search area. They also classified the surveyed coverage approaches according to Choset's taxonomy, including no decomposition, exact cellular decomposition, and approximate cellular decomposition. Their review also considered different shapes for the search area, such as rectangular, concave, and convex polygons.

Generally, multiple robot approaches add a layer of complexity to CPP. The most notable challenge that arises is collision avoidance. Robots need to cooperate to achieve coverage while not only avoiding collisions with obstacles, but also with each other. In 2020, Zhang et al. [36] performed a comprehensive survey on cooperative path planning for UAV groups. They proposed a taxonomy that classified cooperative path planning problems along three axes, namely the type of task, the planning framework, and the environment. The type of task is classified as a rendezvous task, an allocation task, or a coverage task. The planning framework is classified as centralised, decentralised, or hybrid. The environment is classified as known or unknown.

The following sections will explore different types of CPP techniques. Section 2.5 will cover single robot CPP. Single robot path planning will be divided into exact methods, sampling-based methods, A* and wavefront-based coverage, spanning tree coverage, and artificial intelligence methods.

Section 2.6 will cover multi-robot coverage path planning (MCPP). MCPP methods can be classified as distributed or non-distributed, and offline or online. Distributed methods are methods where the paths of individual UAVs do not cross. The search area is typically divided into a number of separate sub-regions and each UAV is allocated its own sub-region to search. Non-distributed methods are methods where the UAVs are free to cross paths. The search area is not divided and the paths of the UAVs are computed simultaneously, with knowledge of which cells have already been visited [25]. Offline

methods are where the path planning is performed and completed before the path execution begins, typically in known environments. Online methods are where the path planning and execution are performed in tandem, typically in unknown or partially-known environments.

2.5 Single Robot Coverage Path Planning

Single robot coverage is discussed in some detail here because several of the MCPP problems make use of them. Distributed MCPP, for example, tends to divide an environment into sub-regions that can then be covered by single robot coverage methods. Sections 2.5.1 and 2.5.2 discuss exact and sampling-based coverage techniques, while Sections 2.5.3 to 2.5.5 cover different grid-based methods.

2.5.1 Exact Methods

Combinatorial methods, as described by Lavalle, are also referred to as exact methods [34]. Exact methods for CPP make use of the same geometric principles to divide an area into cells. However, instead of creating a road-map, an adjacency graph is created and used to move between cells. Each cell is then individually covered, generally using simple manoeuvres [42].

Each cell in the decomposition is a node in the adjacency graph. An exhaustive walk is used to ascertain the sequence in which to visit these nodes to achieve coverage. Simple manoeuvres, such as back-and-forth motions, are then used to cover each cell individually, generally providing complete coverage. [44]

A popular exact method, that is mentioned in Lavalle's book, is the trapezoidal decomposition [34]. This method decomposes an environment into trapezoids (convex cells) based on the vertices of polygonal obstacles. The boustrophedon method builds on the trapezoidal method. It reduces the number of cells by only looking at vertices where a line can extend both upwards and downward from it [42]. This reduces the final length of the coverage path and makes it more efficient.

Both these decomposition methods are applicable in two-dimensional coverage problems. They are offline approaches, since the environment must be known a priori, and only work with polygonal obstacles [42]. This means some approximations may need to be made to represent the environment using polygons.

A more versatile exact method, that uses Morse functions for the decomposition, is also available [45]. This no longer requires polygonal environments and can in theory be expanded to higher dimensional environments.

2.5.2 Sampling-Based Methods

Sampling-based methods have been adapted for coverage path planning. They are more easily scaled to three-dimensional environments and are better suited to online or real-time approaches. They also deal with changing environments containing dynamic obstacles more easily. [34]

Nourani-Voutani et al. [46] used sampling-based CPP to perform automated lawn mowing. RRT was used as a local planner in combination with a global planner that uses a spiral motion to cover the points in a map. A solution is however, not guaranteed. Complete coverage is also not necessarily achieved because of the random nature of the paths, but it is considered a real-time approach.

Englot and Hover [47] used probabilistically complete sampling-based CPP to perform inspection of complex structures. A redundant roadmap algorithm constructs a roadmap and then uses RRT for local point-to-point planning, which also incorporates collision avoidance.

Danner and Kavraki [48] applies a similar strategy, also intended for the autonomous inspection of three-dimensional structures. The method used is, once again, probabilistically complete. A method similar to a probabilistic roadmap is use in the three-dimensional case to achieve coverage.

Wzorek et al. [31] use a combination of PRM and RRT to develop an online point-to-point path planner with replanning capabilities. This was later expanded to a coverage solution, that uses back-and-forth motions, by Rudol and Doherty [30].

2.5.3 A* and Wavefront Based Coverage

A* is a discrete method of planning that is often used in point-to-point path planning. In combinatorial motion planning or multiple-query sampling-based methods such as PRM, road-maps are generally formed to represent the environment. These road-maps can then be navigated using A* or another discrete algorithm. A* was built from Dijkstra's algorithm, which can be seen as a forward search that takes cost into account for the priority queue. A* simply goes on to predict the cost to reach the goal using a heuristic. Dijkstra has also been optimised into what is called a wavefront planner. With this technique, equal cost points are grouped together into "waves" and the algorithm essentially propagates out the waves until it reaches the goal. [34]

Barrientos et al. [49] adapted this wavefront type planning for CPP, with the goal to minimise rotations and the number of revisited cells.

Some authors have taken to extending A* algorithms to CPP as well. Viet et al. [50] combine the A* method with the boustrophedon method, which is generally used for exact CPP. It is an online method that constructs boustrophedon regions incrementally and uses A* to move from one region to the next.

In point-to-point path planning, the goal is usually to achieve the shortest path possible and the heuristic function is set up for that purpose. For CPP, the cost function can be changed to maximize coverage instead. Le et al. [51] use this technique in a grid-based, offline approach where they try to minimise the amount of cells that get revisited. They use critical way-points and A* based zigzag motions.

Dogrue and Marques [52] use a heuristic function with the goal of minimising the number of rotations, similar to the wavefront planner mentioned earlier . This can be useful, because rotations often consume more energy than straight-line motions.

2.5.4 Spanning Tree Coverage

Spanning trees are applied in discrete environments. When they are used in a coverage path planning application, it is referred to as spanning tree coverage (STC), which can be used in an offline or online approach.

Gabriely and Rimon [53] show the various ways in which spanning trees can be used to achieve coverage. They first show the offline case where the environment is known in full prior to the planning phase. The environment is discretised into a grid of large cells which each consist of four smaller cells. The robots traverse the smaller cells, but the large cell centres are used as the nodes for a spanning tree formulation. The spanning tree can then be circumnavigated to achieve approximately complete coverage without any backtracking. A minimum spanning tree (MST) algorithm, called Prim's algorithm, is used to create the spanning tree. This minimises the total weight of the tree. The weights can be used to favour a certain coordinate direction for searching.

An online STC technique is also shown, where the only prior knowledge of the environment is that the obstacles are static. The algorithm grows the spanning tree similar to with the offline approach, but does so incrementally as more knowledge of the environment becomes available.

Because of the circumnavigation method, the coverage paths generated in the offline case are closed-loop paths. In the online case, the map is generally growing outwards from the starting location, generally resulting in a coverage path with one or several spiral shaped sections.

2.5.5 Artificial Intelligence Methods

Juan et al. [25] compared several AI techniques for CPP. Four methods were compared, including one that employs a GA. The four methods are the La Palma attraction, La Palma fuzzy logic, adaptive-network-based fuzzy interference system (ANFIS) and particle swarm optimization (PSO) approaches.

All of the methods were implemented in a discrete, gird-based environment. A risk/occupancy map is given as an input to each environment. This

map assigns a priority to the cells that encourages covering of certain areas first.

The performance of these algorithms was evaluated in the context of SAR, and it was found that the ANFIS approach gives the best performance for this application. If there is little variation in priorities between cells, the attraction method works well. If a big section of the environment has high priority values, the fuzzy logic approach was found to work well. In general the PSO technique did not perform well.

2.6 Multiple Robot Coverage Path Planning

This section discusses coverage path planning (CPP) when multiple robots are used. Offline techniques are discussed for the distributed case in Section 2.6.1 and for the non-distributed case in Section 2.6.2. Online techniques are briefly discussed in Section 2.6.3.

2.6.1 Distributed, Offline Methods

A well established offline CPP approach involves the divide areas technique. This partitions an area into regions for individual robots to cover. Each robot should then be able to cover its area using a single robot coverage path planning technique.

A number of different area division approaches are discussed in this section. The methods used to perform coverage of the sub-regions are mentioned in each section seen as these are important in generating the final coverage plan. Figures are shown in these sections, all of them taken from the research papers discussed in the respective sections. These are to illustrate the different divisions that are achieved by each algorithm.

2.6.1.1 Hexagonal Segmentation

Azpúrua et al. [2] formulated a distributed MCPP approach use with geophysical surveys with UAVs. Their implementation uses regular hexagons to segment the area of interest. The hexagonal cells are equal in size and are each assigned to a single robot for searching.

Hexagons are clustered using the K-means algorithm in order to assign them to robots. This ensures a similar number of cells are assigned to each robot. The seeds are synonymous with the robots, therefore once the seed locations are finalized for even cell distribution, the robot initial positions are established. The resulting sub-regions, each consisting of a number of hexagonal cells, are contiguous. All the hexagonal cells assigned to one robot can then be covered using back-and-forth motions in a similar way to exact single robot coverage methods.

Having similar sized sub-regions means that each robot can execute its coverage path in a similar amount of time, which is an advantage when optimizing for fuel usage and mission completion time. However, this algorithm does not allow for random robot initial positions, which can be a disadvantage.

Static obstacles are considered in this implementation, but the smallest obstacle resolution is the size of a hexagon which may not be very representative of the environment. Coverage is however resolution complete, with the hexagon size representing the resolution.

The application was tested in a real-world environment using UAVs with a limited flight time. The paths were found to be feasible, even with the presence of sensor noise and environmental factors.

Figure 2.2, taken directly from their paper, illustrates the back-and-forth manoeuvres used to cover the hexagonal partitions. Black hexagons represent no fly zones and/or static obstacles. The dark red, green and blue regions represent the sub-regions as they are assigned to the respective robots for coverage.

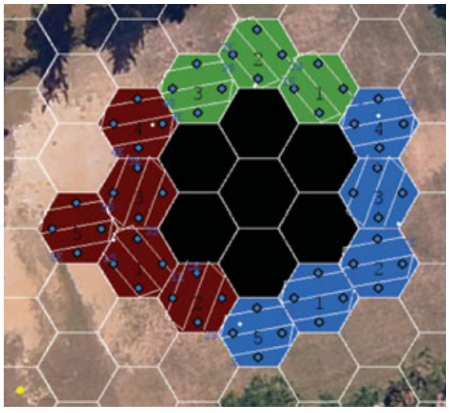


Figure 2.2: Simulation showing coverage of hexagonal partitions with back-and-forth motions using three robots. [2]

2.6.1.2 Voronoi Partitioning

Nandakumar and Rao [54] show a method of area division to divide a polygon into a number of equal area polygons. Another relevant method that also stems from the field of mathematics, is the Voronoi partition. This assigns regions within an area to seeds based on distance. The idea is that a region assigned to a seed represents all the points where the distance to that seed is shorter than to any other seed.

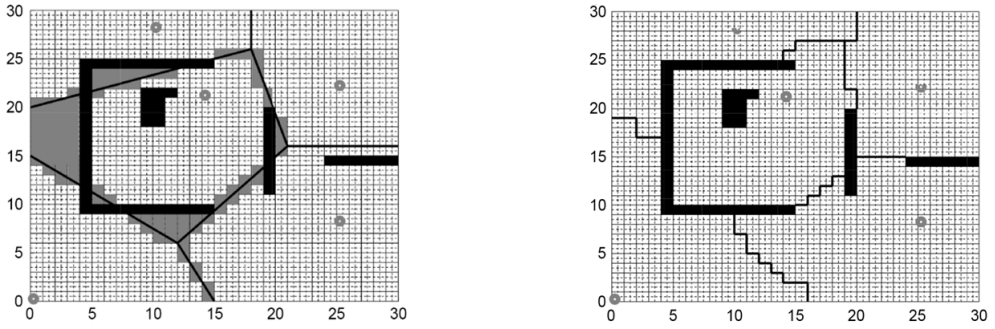
If the Voronoi partition is applied to the MCPP problem, the seeds become synonymous with robots. This partition works for any number of robots at any starting positions, but unless they are evenly spaced, the areas will not have equal sizes. Distances in these scenarios are usually Euclidean and the

boundaries between areas represent the position where the distances from two seeds are equal.

Nair and Guruprasad published an article in 2020 [3] that implements MCPP using Voronoi partitions in discrete space with static obstacles. They use a grid-based representation of the area and compare several different methods. They investigate geodesic-Manhattan-, Manhattan-, geodesic- and Euclidean-distance-based Voronoi partitions.

The Euclidean-based technique results in what the authors term "non-contiguous sub-regions". This means that cells that are part of a sub-region are not accessible by the robot assigned to them, due to obstacles within that sub-region. They solve this problem by using geodesic distances. This uses Euclidean measurements, but instead of a straight line distance between two cells, it calculates the distance using a collision free path between the two cells.

Another problem arises, due to their use of discrete space. When using Euclidean distances, some cells were partially in two sub-regions instead of fully in one or the other. Their solution to this is to use Manhattan distances. Ultimately they make use of geodesic-Manhattan-based distances to generate the partition. And thus they coined the term geodesic-manhattan voronoi-partition-based coverage (GM-VPC)).



(a) Euclidean (b) Geodesic-Manhattan

Figure 2.3: Illustrations showing results for the Voronoi partitioning scheme for two different distance measures. [3]

Figure 2.3 shows figures from the article that show the result of an area division using different distance measures with a Voronoi partition. In both figures, the black blocks represent obstacles, the round dots are the robot starting positions and the black lines over the grid represent the Voronoi partition boundaries. In Figure 2.3a, the grey blocks are areas that would not be covered. This is clearly remedied using the GM-VPC technique shown in Figure 2.3b.

They tested these partitions in simulations with exact and an approximate individual area search techniques. They implemented a boustrophedon

coverage plan for the exact solution and a spanning tree for the approximate version. Both performed better when using geodesic-Manhattan distances. [3]

2.6.1.3 Negotiation Protocol

A negotiation or bargaining protocol refers to a process involving task partitioning. In the context of area division for CPP, the task represents the area to be divided.

Rossi et al. [4] presents a negotiation model using Rubinstein's alternate-offers protocol, for the purpose of area division. Barrientos et al. [49] furthered this implementation by introducing a wavefront planner for the individual area coverage technique. A series of field tests were also done to evaluate the system's performance in the context of precision agriculture.

The focus of this implementation was to develop a distributed algorithm capable of considering robot capabilities. This means that the robots would not have to be homogeneous and can have different flight-time capabilities, manoeuvrability, on-board equipment and so forth.

They implemented their algorithm and found that it can achieve near optimum results. It tries to maximise the size of each robot's subdivision of the area (based on its capabilities), while also minimising sub-area overlap. The algorithm also works to avoid static obstacles or no fly zones in the area. Figure 2.4 shows an example of an area division achieved using this method. The area is divided into a red and a green region for two different robots. The blue region represents a no-fly zone.

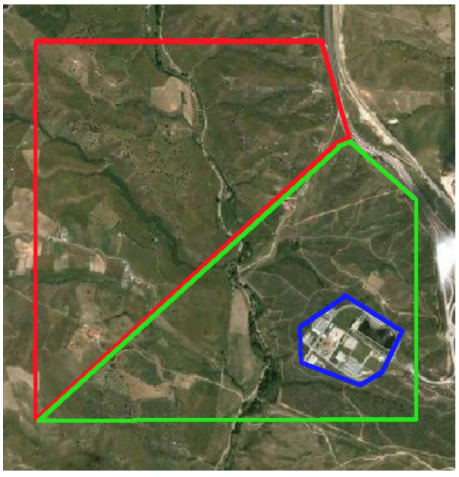


Figure 2.4: Illustration of the resulting area partition using the negotiation protocol. This example is for two robots and includes a no fly zone. [4]

After area division, the environment is discretised into cells based on the onboard camera field of view (FOV), so that the wavefront planner can be

used to cover these cells. In order for the polygons generated by the negotiation protocol to work effectively, they used a method called Bresenham's line algorithm to approximate the lines that divide the areas in discrete space, so that they pass through the cell centres.

The area division achieved sometimes produces non-convex shapes, which the wavefront planner can handle effectively. Their implementation also minimizes energy consumption by minimizing the number of rotations and not allowing backtracking. In addition, they have the ability to specify the initial take-off positions of the robots. Distance from the specified take-off point to the starting point for sub-area coverage are considered in the sub-task negotiations. It is also mentioned that they are able to specify robot landing positions pre-emptively.

One visible drawback in their implementation is that the coverage appears incomplete. The boundaries between areas pass through way-points (cell centroids), that effectively get excluded from the coverage algorithm and are not covered.

2.6.1.4 Multi-Robot Spanning Tree Coverage

Multi-robot spanning tree coverage (MSTC) is a variant of single robot STC. Hazon and Kaminka [5] published an implementation of MSTC in a 2005 paper. Two variations of MSTC were given; one that allows for backtracking and one that does not. Both variations still utilize a single spanning tree, but simply circumnavigate the tree with multiple robots instead of a single one.

They placed emphasis on robustness and efficiency, in addition to completeness. They demonstrate an algorithm that segments the path around a spanning tree to evenly distribute it among robots. Their method becomes inefficient, however, when robots are clustered closely together. This is because a robot simply navigates the path until it reaches the initial position of the next robot on the path.

Figure 2.5 shows the paths that are generated when the robots are evenly distributed along the path that circumnavigates the tree. Blue dots represent the robot initial positions and the spanning tree is shown in red. The second method they suggest remedies this somewhat. It allows for backtracking and improves the efficiency.

The ideal situation is that all the robots have near equal path lengths, provided they are homogeneous robots. This is not guaranteed with this algorithm when the robots have random starting positions, but allowing for backtracking can improve the results and allow the coverage to be completed in a shorter amount of time.

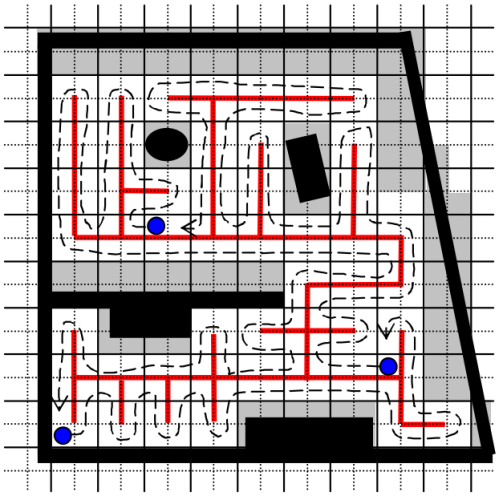


Figure 2.5: MSTC algorithm showing the paths for three robots on an environment grid. [5]

2.6.1.5 DARP

Kapoutsis et al. [6] formulated a unique distributed technique called the divide areas algorithm for optimal multi-robot coverage path planning (DARP). This algorithm divides an environment amongst multiple robots based on their starting locations within a grid-based environment. It makes use of an iterative approach that causes the sub-regions to tend to an optimal division over time. The environment includes static obstacles, but obstacles that form enclosed, unreachable spaces are not permitted.

The algorithm starts by assigning each cell to a UAV based on which UAV is the closest to it. The algorithm then adjusts cell distance values iteratively to change cell assignments and formulate a solution. An optimal solution is achieved if all cells are assigned to only one robot, all the sub-regions are the same size, the sub-regions are contiguous, and the UAV that is assigned to a specific region has its initial position in that region.

Equal-sized sub-regions means that the time to cover each sub-region would be similar, provided the UAVs are homogeneous. If the regions are contiguous and each contain one UAV initial position, it means the regions can be covered independently, without the UAVs needing to traverse each other's regions. This means that collisions between UAVs are eliminated.

The results of the area subdivision achieved by DARP can be seen in Figure 2.6, which is a graphic taken directly from their paper.

A spanning tree coverage (STC) method was used to cover the individual sub-regions, resulting in complete coverage at the grid resolution. The performance of the algorithm was measured against the MSTC and multi-robot forest coverage (MFC) methods. A comparison is made between the algorithms to demonstrate which performs best for creating paths of equal length for each

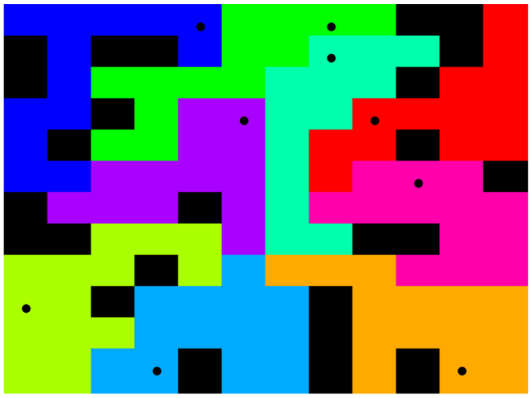


Figure 2.6: Illustration of the area division achieved on a grid with static obstacles, using DARP. [6]

robot. DARP was shown to out perform both other algorithms, with at most a discrepancy of four cells between the longest and shortest robot coverage paths.

Several other authors have made use of this algorithm to date. Gao et al. [55] applied ant colony optimization to the DARP and STC combination to reduce the number of rotations, thereby reducing overall energy consumed during flight. Baras et al. [56] addressed the unreachable region problem by allowing for vertical manoeuvres to avoid obstacles. Generally, their algorithm operates the same as the original DARP algorithm, but adds a second phase where unconnected regions are handled using three-dimensional manoeuvres.

2.6.2 Non-Distributed, Offline Methods

This section covers three methods that cannot be classified as distributed because the planned coverage paths may cross. The area is not divided into sub-regions as part of the planning process. Different techniques are covered in each subsection. Interesting to note is that these methods generally adapt existing single robot coverage techniques.

2.6.2.1 MCPP Using MFC

Zeng et al. [57] published the multi-robot forest coverage (MFC) as a multiple robot coverage technique in 2005. The intent was to improve upon the multi-robot spanning tree coverage (MSTC) method. Their idea was to construct a tree with the consideration that it will be divided afterwards, unlike what MSTC does. It allows for robot path overlap, which means there is redundant coverage and collision avoidance amongst robots would need to be considered. However it can handle unique scenarios, where backtracking is unavoidable, quite well.

Even et al. [58] published an article on the approximation algorithm that was used for the development of MFC, which uses the rooted tree cover scenario in particular. For MFC, the roots represent the robot initial positions and then a tree is generated for each robot. An objective to minimize the weight of the maximum weight tree is also applied. These trees are each circumnavigated by their robot (root) to cover the area.

Based on simulations with MSTC and MFC, they found that MFC generated closer to optimum results and generally achieved coverage in a shorter amount of time. Because there is path overlap in certain scenarios, MFC is not a truly distributed method. Results generated by this algorithm can however resemble a distributed algorithm quite closely, depending on the environment.

2.6.2.2 MCPP Using Artificial Intelligence

Juan et al. [25] used artificial intelligence (AI) methods for the single robot CPP case, but also extended their application to the multiple robot CPP case. They investigate the distributed case, however they do not present their method for dividing the environment into sub-regions. They also investigate the free formation case for the two and three UAV scenario, which will be the main discussion of this section.

Free formation CPP means that the paths for multiple UAVs in an environment are planned simultaneously, with knowledge of which cells have already been visited by the robots. This means that the robot paths will potentially cross in the environment, meaning that collision avoidance would need to be considered for real-world implementation. They do not address collision avoidance in their article though, and simply allow the paths to cross.

A risk/occupancy grid is applied to the environment once a priori. This encourages the algorithms to visit certain regions of the map first, by assigning priorities to the cells. The algorithm is designed to be used in a SAR scenario, where this would be useful.

Three AI methods were investigated for free formation MCPP in a grid-based environment with priority assignments. The La Palma attraction method was found to produce the shortest paths for environments with fairly homogeneous priority assignments. The adaptive-network-based fuzzy interference system (ANFIS) method comes in close second, and the fuzzy logic approach produces significantly longer paths. The fuzzy logic approach tends to perform well in regions of the environment with relative high priority values. Overall, the ANFIS method had the best performance over a range of priority grids.

2.6.2.3 MCPP Using Linear Programming

Linear programming can be used to optimize linear problems with a number of variables, by trying to minimise or maximise some cost. They are also usually subject to several constraints.

Avellar et al. [59] adapted this methodology to an application with multiple UAVs for CPP. They designed their optimization problem with the main objective of minimising coverage time. Their algorithm does this by minimising the length of the longest UAV flight path.

One contribution they provide is the consideration of setup time. By their definition, setup time refers to the time taken by an operator to prepare a UAV for flight. They specifically consider scenarios where setup times can accumulate due to there being less operators than UAVs, which leads to an accumulation of setup time for some UAVs.

Several constraints are applied to the problem. They limit the flight times of UAVs based on battery power, set a constraint so that every node can be visited by only one UAV and limit the paths to closed-loop paths.

To ensure complete coverage they develop a constraint that ensures all nodes in one row are visited by one UAV. They also have two optional constraints to avoid diagonal lines that cross the environment. Obstacles are not considered in their implementation, nor are UAV collisions.

2.6.3 Online Methods

Online path planning generally refers to scenarios where a plan is generated while information about the environment is still being collected. This is often applicable in highly dynamic environments, where obstacle positions are difficult (or costly) to predict a priori. This section discussed Online MCPP briefly.

Some of the single robot coverage algorithms have online versions. Viet et al. [50] used boustrophedon-A*, where the boustrophedon regions of an environment are constructed incrementally and A* is used to move from one region to the next for coverage. Gabriely and Rimon [53] formulated an online spanning tree coverage (STC) algorithm, by growing the spanning tree incrementally as the environment becomes known.

Sampling-based methods are well suited to online approaches. Single query approaches like RRT avoid explicit representation of the environment and can therefore be used in dynamic environments more easily [34].

Often algorithms also use a hybrid of online and offline, where some information about the environment is known a priori, but data is still collected to update aspects of the environment incrementally. Often, because of the online element in the algorithm, it is still considered as online overall. [34]

The paths of the UAVs are generally created dynamically for online path planning, and because all the environment information cannot be known a priori, it is not generally possible to guarantee complete coverage. [34]

When it comes to online CPP with multiple robots, there are a few examples. Luo and Yang [60] published an article showing CPP for multiple cleaning robots . This application implements a neural network to plan paths for multiple robots in a dynamic environment. No learning procedures were

executed for this algorithm. The robots treat each other as dynamic obstacles within an environment and are at all times aware of the other robot positions relative to themselves. The goal is also to minimise rotations and avoid collisions with other obstacles and robots in the environment, while also covering the whole area.

They show that the algorithm works effectively in a warehouse environment with ground vehicles, and gives real-time performance. The robots cover the entire area while avoiding collisions with each other and obstacles, and without ever crossing paths or backtracking.

Waharte and Trigoni [29] developed an online application with multiple UAVs, intended for search and rescue operations. Their algorithm iteratively updates an occupancy grid of the environment. This represents the likelihood of the target being in any given grid cell, based on the information collected. This information is used by each UAV to choose its next action using a steepest gradient method. This approach can arguable be considered to favour target finding over achieving coverage, but the result is similar to that of a coverage algorithm.

Hayat et al. [32] used a GA to optimize coverage and communication with multiple UAV in a SAR operation. Connectivity to the ground station can be prioritised to ensure new information about the target can reach the SAR teams efficiently. This information can be used for dynamic re-planning of UAV paths. A multi-objective path planning (MOPP) algorithm was used to favour connectivity and coverage in various degrees. For a smaller group of UAVs, favouring connectivity was found to be better whereas favouring coverage gave better results for a larger group.

2.7 Key Findings and Design Decisions

This section summarises the key findings from the literature, as well as the research decisions that were made based on the knowledge gained.

Based on the literature it was concluded that UAVs can provide valuable support for SAR operations. An aerial search using UAVs is, however, not well suited to any type of environment. SAR operations are divided into a number of categories based on the type of environment. An aerial UAVs search is well suited to ground, mountain or maritime SAR. It is however not appropriate for cave SAR and is not well suited to combat and urban SAR.

During the planning stage of a SAR operation, probable survivor locations are identified and the available resources are allocated to various search areas. UAVs may be used as part of a larger SAR operation. A search coordination team may demarcate a sub-area of a larger search region for searching by UAVs.

If this UAV search is automated, it would mean a search can be conducted without a significant increase in human resources required. One area can

also be covered faster by multiple UAVs when compared to a single UAV. Using multiple UAVs does however mean that consideration must be made for collisions between UAVs.

When modelling a multiple UAV automated SAR operation as a coverage path planning (CPP) problem, one has the option of distributed or non-distributed solutions. Distributed CPP intrinsically provides collision avoidance because individual UAVs search non-overlapping sub-regions of the environment independently of one another. Non-distributed solutions allow for paths to cross which means that collision avoidance would need to be considered.

Online path planning provides the flexibility of adapting to dynamic environments that cannot be known in full prior to a search. Offline path planning is however suitable for environments that can be known a priori.

The goal of this research is to develop an automated SAR approach that uses multiple UAVs to cooperatively search a demarcated area. An aerial search method is considered most appropriate and it will only be employed for ground, mountainous and maritime environments.

The automated search will be modelled as a CPP problem so that the UAVs perform a systematic search of their allocated search area. A distributed method will be used to eliminate collisions between UAVs. The environment is typically known prior to a SAR operation, so the problem is modelled as an offline one with static obstacles.

Since SAR operations typically involve large areas, fixed-wing UAVs are assumed to be more appropriate for conduction the search. They tend to have longer endurance than rotary-wing alternatives.

Chapter 3

Conceptualization and Modelling

The main goal of this chapter is to outline the assumptions made for the purpose of this project, and what limitations they impose on the application. A general mathematical model of each component in the system is established to provide a model of the overall problem. Section 3.1 illustrates the general SAR problem, followed by Section 3.2 which describes the environment and how it is modelled. Sections 3.3 through 3.7 describe and model the elements within the environment, including obstacles, UAVs, and the target that needs to be found.

3.1 The SAR Problem

Aerial support in a SAR operation can be very useful. Historically, manned aircraft like helicopters have been used to do so. However, these manned aircraft have limited flying times and their flight paths are not necessarily optimal for detecting a target in a SAR situation.

There has been some use of robots, including UAVs, to assist in SAR. Section 2.2 detailed some of these uses. UAVs bring an exciting opportunity to have vehicles supporting an operation with autonomous flight paths. Search paths can be optimized to reduce the time to find survivors.

Moreover, no-one is required to fly the vehicle, not only reducing the danger to a potential pilot, but also allowing the manpower to be focused on locating and extracting survivors. Often certain areas may be difficult to reach and require specialised teams to search these regions for survivors [19]. Unmanned vehicles may be useful to search these areas, without endangering a search team. A combination of manned an unmanned vehicles may also be beneficial to find and aid survivors faster in a SAR operation.

DroneSAR, which was discussed in Section 2.2.2.1, proved that the use of UAVs can significantly speed up a SAR operation and assist in target finding. They managed to find a target in a one square kilometre area within 20 minutes

when assisting with one autonomous UAV. In contrast, a ground team of five people took about two hours to achieve the same goal without assistance [1].

One can deduce that using multiple UAVs to assist a SAR operation would improve the time to find survivors even more. If they search in tandem, larger areas can be searched in shorter amounts of time. Using a multiple UAV, systematic search of an area as a starting point, one can formulate the basis for a SAR problem.

In Figure 3.1 one can see all the basic components of a UAV-assisted SAR operation illustrated and labelled.

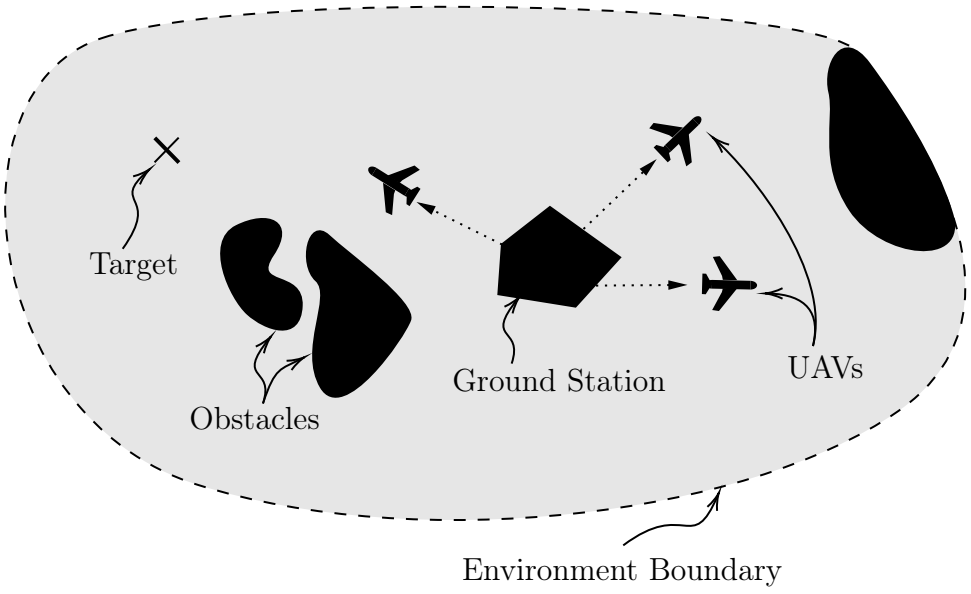


Figure 3.1: Diagram showing the components of an automated Search and Rescue problem with multiple UAVs.

The first noticeable component is the search region boundary. This represents the area that is demarcated by a rescue team for searching. Within this region is the target that needs to be located. This is marked by an "**X**" on the diagram, and represents a survivor or group of survivors that need rescuing.

A distinction should be made between the entire area to be searched, and the area that is to be systematically searched by the UAVs. In Section 2.1.2, the different stages of SAR are described. The stage in which the entire search area is specified and a search plan is formulated is the planning phase [19].

It is assumed that this would be the phase wherein it would be decided whether to use a systematic UAV search as a means to assist a SAR operation. Systematic searches of the entire search area are uncommon in SAR, because the areas that need to be searched are often large and this would be impractical. They often choose a probabilistic approach, starting in regions where the survivors are more likely to be [19].

If a systematic search is deemed impractical over the entire area, UAVs may still be useful. Instead of using UAVs to search the whole area, they may be used to search one or multiple demarcated sub-regions within the area, as part of a larger search plan. The actual search would then be conducted in the operations phase [19].

It is important to note is that a search may be abandoned if it is considered too dangerous for a search team [19]. Using automated UAVs instead of manned aircraft or search teams means that a search could continue even when it is too risky for search personnel to conduct.

In most rescue scenarios, the search team can make an educated guess as to where the target may be. The UAVs therefore do not know the exact location of the target prior to flight, and are in fact trying to pinpoint their location. The UAVs are also shown on the diagram as they are leaving the ground station.

It is intuitive to have a base of operations for UAVs during a SAR operation, seeing as this would be where they take off and land, as well as where they refuel if necessary. This base station would also most likely be from where the search team receives any data collected by the UAVs to assist in the SAR operation.

Static obstacles within the environment are also shown. These are representative of any region wherein the UAVs must not fly. These could be physical obstructions, such as power lines or cliffs, or no-fly zones such as populated areas or restricted airspace. The UAVs must fly throughout the unobstructed air space until the target is found.

3.2 Search Environment

SAR scenarios can be classified according to the type of terrain the search is conducted in. In general, SAR is divided into ground, urban, mountain, cave, combat, or maritime rescue [21]. Section 2.1.3 discussed the distinctions between these different kinds of SAR operations.

The extent of assistance that automated UAVs can provide would depend on the type of rescue. In wide open plains or oceans with few obstructions, they would have a clear view of the search area from any visual or thermal camera. In densely forested areas, visibility may be limited due to vegetation, and aerial support would be unhelpful.

Unfavourable weather conditions also challenge visibility and flight capability. UAVs are generally designed to fly in certain wind conditions. High wind speeds could make it impossible to fly. Snow, rain, fog, lightning or dust may also impair the flying capabilities of a UAV by damaging components or making sensor readings inaccurate. High humidity in particular has been known to impair optical sensors. [61]

Aside from impairing components, these elements also reduce visibility and make it more challenging for cameras on UAVs to detect targets. At night UAVs may have to rely fully on thermal images, since the image on a conventional camera would by mostly dark.

UAVs would also face challenges in extreme temperatures. Areas with fires would have to be considered no-fly zones for UAVs. Areas where temperatures are either too high or too low for the components would also need to be excluded from the search region.

As mentioned in Section 2.1.3, low density air in mountainous regions can also be a hazard. In general this means that fixed-wing UAVs would be better suited for this kind of search. It should be noted that mountain rescue search teams often work in difficult terrain and need specialised skills and equipment [21]. Using an aerial search could reduce the risk to these kinds of teams.

Cave searches are also generally grouped with mountain searches. However, a high-flying, systematic search with UAVs would not be functional in caves. For this reason, caves are not considered for this research. Combat rescue is also not addressed in this project, seeing as the circumstances of a battlefield are volatile and unpredictable compared to the other scenarios, and would require unique consideration.

Natural obstructions can become obstacles to UAVs depending on the altitude at which they fly. These could be trees, rocky outcrops, or any other natural elements tall enough to get in their way. Man-made obstacles like telephone towers, power lines, towers and bridges could also be obstacles during flight [19]. Man-made structures and debris can also cause visibility challenges for an aerial search, if survivors are in or under structures.

Urban search and rescue, where some kind of disaster has occurred, generally involves people trapped in rubble of some sort. A systematic aerial search may therefore not be quite as useful for locating survivors. However, they have been used in the past for large scale disasters like Hurricane Katrina. This was discussed in Section 2.2.1.

Flying over populated regions poses unique challenges. It would be more useful if this were considered independently, with UAVs serving in a role other than target location, such as surveying. For this reason, urban SAR is not considered specifically in this project.

There may also be obstacles in the form of no-fly zones. These could be restricted airspace, populated areas, private land or protected wildlife areas wherein UAVs are not permitted to fly.

In general UAVs suffer from similar limitations that manned aircraft would, and it is assumed that they would be used with careful consideration by a search team. The advantage is that if a UAV flies in dangerous conditions, no pilot is placed in danger and the worst case scenario involves a crashed UAV.

The area wherein the UAVs are expected to assist the search operation is assumed to be a bounded geographical region. In the larger search plan,

the region that needs to be systematically searched by UAVs needs to be demarcated. The environment is represented in two dimensions, seeing as the UAVs are expected to fly at a constant altitude. This will be discussed further in Section 3.4.

The area within the boundary of the demarcated region is referred to as the environment ($\mathcal{E}$) for the search. This environment should be fully mapped prior to the systematic search that needs to be executed by the UAVs. An example of a map for an environment that has been demarcated for search is shown in Figure 3.1.

The environment can be seen as a set of continuous points within a certain coordinate bounds. It should be noted that any point within this map should be reachable by a UAV from any other point in the map, not including points where there are obstacles. There should be no enclosed regions. Chapter 5 will discuss, in more detail, how an environment can be modelled, for the purposes of this project, using tools such as digital elevation models (DEMs).

The UAVs and obstacles are located within this environment. It is possible that the survivors are located outside of the region being searched, but this does not drastically alter the implementation. The UAVs would simply complete their search without locating the target of the larger search.

3.3 Environment Obstacles

The environment is assumed to be static, and dynamic obstacles are not modelled explicitly. Dynamic obstacles at higher altitudes are unlikely. In regulated airspace, flight plans and collision avoidance by other aircraft means that the UAVs will most likely not have to avoid other aircraft. The assumption of a static environment that can be mapped in full prior to the search is a limitation of this implementation, but at higher altitudes it is considered to be a valid one.

A short-term, online collision avoidance system, similar to the one developed by Meiring et al. [16], is assumed to be onboard the UAV. This system would help UAVs avoid collisions with dynamic obstacles, such as birds, that may be present in the environment. This onboard collision avoidance would be designed to adhere as closely as possible to the original path, thereby not having a noticeable effect on the flight time and energy consumption of the UAVs overall.

Different types of obstacles are discussed in Section 3.2, since they tend to be environment specific. Obstacles can include man-made structures, natural obstacles and no-fly zones. These no-fly zones could be due to treacherous weather, inaccessible terrain or restricted airspace. No-fly zones could also simply be areas that have already been searched. A search team may also exclude areas that they are confident the survivors are not occupying, or areas that are being searched using some other strategy instead of UAVs.

The set of points in an environment that are determined to be obstacles ($\mathcal{O}$) is a subset of the environment space. The search region ($\mathcal{S}$), that needs to be systematically searched by the UAVs, is therefore the environment excluding this set of obstacles. This is mathematically expressed as

$$\mathcal{O} \subset \mathcal{E} \tag{3.1}$$

$$\mathcal{S} = \mathcal{E} \setminus \mathcal{O} \tag{3.2}$$

where $\subset$ is the subset operator, and $\setminus$ is the set subtraction operator.

3.4 UAV Model

In order to simplify the systematic search, the UAVs are assumed to all fly at the same constant altitude. They are also assumed to be homogeneous, in that they are all the same model of UAV and have the same limitations. This makes the environment planar, or two-dimensional.

The starting point for a UAV for the systematic search would be a point within this planar environment. This starting point is referred to as the initial position of a UAV. This is defined as the position of the UAV after it has already reached the required altitude and ready to start the systematic search. Two manoeuvres are required to reach this initial position from the ground station. The first is referred to as take-off, which is the manoeuvre required to reach the search altitude. The second is referred to as departure, which is the manoeuvre necessary to reach the initial position of the UAV.

The initial state of the UAV has a heading (ψ_r), as well as a position represented by a two-dimensional coordinate. The UAV is represented by a point mass, therefore the two-dimensional coordinate is indicative of the position of the UAV's centre of mass. The heading can also be described as the yaw of an aircraft. The aircraft is expected to have an associated roll angle (ϕ_r), but this is excluded from the state since it would not affect the planar heading. The pitch angle (θ_r) is also excluded, since it is assumed that there are no altitude changes during the search.

The initial state of a UAV is expressed as

$$X_{\mathrm{I}} = (x_0, y_0, \psi_0) \tag{3.3}$$

where X_{I} is the initial state of the UAV, x_0 and y_0 are the two-dimensional coordinate describing the position of the UAV, and ψ_r is the heading of the UAV at its initial position. The position of the UAV is constrained by

$$(x_0, y_0) \in (S) \tag{3.4}$$

where $\mathcal{S}$ is the entire search area.

From the initial states, a search path must be generated for each UAV. The final state of the UAV is then considered to be the last point in this path, with the associated heading of the UAV at that point.

After reaching this final state, an approach and landing manoeuvre would get the UAV back to the ground station. Landing refers to the altitude change manoeuvre to get from the search altitude back to the ground. Approach refers to any manoeuvre that would be required after the final state of the UAV and prior to the landing manoeuvre.

The search path is a set of discrete waypoints within the planar environment. The planned search path of a UAV is represented by

$$X_P = ((x_0, y_0), ..., (x_k, y_k), ..., (x_m, y_m)) \quad \forall k \in 0 ...., m \tag{3.5}$$

where X_P is the path of the UAV. In this path, (x_0, y_0) is the initial position of the UAV, (x_m, y_m) is the final position of the UAV, and (x_k, y_k) represents intermediate waypoint positions which are constrained by

$$(x_k, y_k) \in (S) \quad \forall k \in 0 ...., m \tag{3.6}$$

where $\mathcal{S}$ is the entire search area.

The UAVs are expected to execute a constant speed search. For this reason, speed is not included in describing the state of a UAV. There may be slight fluctuations in speed for a real-world application. The time scales over which the UAVs accelerate and decelerate are considered to be much shorter that the time scales between waypoints. Therefore, acceleration does not need to be modelled.

The constant speed assumption ensures that when the UAVs have equal length paths, they will also have paths that take the same amount of time to complete. This makes optimizing the search for multiple UAVs easier, and aids in the refuelling calculations. Maintaining constant speed also generally consumes less energy than manoeuvres which require acceleration and deceleration.

Constant speed also eliminates UAV manoeuvres that take time to execute. A multi-rotor, for example, can make sharp turns. It does however need to slow down to execute these manoeuvres. It can also execute a hover, but both these manoeuvres would be excluded from a constant speed implementation. This is advantageous because SAR is a time sensitive application.

A UAV is expected to be able to execute a semi-circular or straight-line manoeuvre, or a combination of the two, when moving between waypoints. When executing semi-circular manoeuvres, there is a minimum turning radius for a constant speed aircraft, which is described as

$$r_{\min} = \frac{V_f^2}{g \cdot \tan(\phi_{\max})} \tag{3.7}$$

where V_f is the constant forward speed of the UAV, ϕ_{max} is the maximum bank angle of the UAV, and r_{min} is the minimum turning radius of the UAV.

Once the continuous paths of each UAV are established for the search, the path lengths can be calculated. The path length is the sum of the lengths of all the individual manoeuvres required to make up the path and can be expressed as

$$P_i = \sum_{k=0}^{m-1}(\ell_k) \quad \forall i \in 1, ..., n_r \tag{3.8}$$

where P_i is the total path length for the i^{th} UAV, ℓ_i is the length of the k^{th} path segment, and m is the number of path segments. Each path segment can be viewed as a UAV manoeuvre.

The trajectory planning will be discussed in detail throughout this project. However, it is assumed that the UAVs have onboard guidance controllers that control the UAVs to follow the planned paths. The continuous paths are converted into a discrete set of waypoints that serve as an input vector to the guidance controller. For the purpose of this project it is assumed that the UAVs can follow their planned paths exactly.

The amount of fuel a UAV can carry constrains the time it can fly. Since the UAVs are homogeneous, they all have the same endurance limitation. In this project, the energy constraint is represented as a predicted flight time (T_p). The search path length for each UAV would be limited according to

$$P_i \leq T_p V_f \quad \forall i \in 1, ..., n_r \tag{3.9}$$

where P_i is the search path length of the i^{th} UAV, V_f is the forward speed for the UAV, and T_p is the maximum predicted flight time of the UAV. If energy consumed during take-off, landing, departure and approach are accounted for the equation becomes

$$P_i \leq (T_p - (T_T + T_D + T_A + T_L))V_f \quad \forall i \in 1, ..., n_r \tag{3.10}$$

where T_T, T_D, T_A, and T_L are the take-off time, departure time, approach time, and landing time, respectively. These are subtracted from the total predicted flight time to get a predicted flight time for the search paths in particular.

These equations assume that all manoeuvres consume energy at the same rate. A safety factor can be applied to semi-circular manoeuvre and other relevant manoeuvres to account for higher energy consumption. This will be addressed later in the project in Section 7.4.3.

When the take-off and landing are modelled explicitly, it is assumed that take off occurs from a central location, or ground station. This ground station is assumed to be chosen by the search team, and is expected to have a single location from which UAVs can take off and land. It is assumed that the ground station location is chosen as a clearing where the UAVs will not experience any collisions during take-off and landing.

It should be noted that a constant climb (V_c) and sink (V_s) rate are assumed for a UAV during take-off and landing. This eases the time calculation

for these manoeuvres. It is also assumed that a UAV starts its departure and ends its approach in the centre of the ground station. The headings for the end of take-off and the start of landing are also the reverse of one another.

3.5 Collisions Model

There are two types of collisions that can occur for a UAV in the environment, namely collisions with the environment, and collisions with one another. A collision with the environment occurs if the UAV's position intersects an obstacle or no-fly zone, which includes leaving the boundary of the environment. A collision between UAVs occurs when two UAVs occupy the same space at the same time.

The assumption is made that the UAVs are homogeneous for a singular implementation. They are all the same type of UAV, implying that they have the same dynamic constraints and overall dimensions.

To model collisions, one starts by representing the position of the UAVs at a particular instant in time. This is expressed by

$$X_i(t) = (x_i, y_i) \in \mathbf{S} \quad \forall i \in 1, ..., n_r \tag{3.11}$$

where $X_i(t)$ is the position of the i^{th} UAV at an instant in time, which is expanded as a two-dimensional coordinate (x_i, y_i). $\mathbf{S}$ is portion of the environment that is free of obstacles, and represents the entire search area.

Because the UAV is represented as a point mass, some provision must be made for the space it actually occupies in order to detect collisions. This can be done using exclusion zones. In essence, the exclusion zone around a UAV or obstacle should be such that if the point mass of another UAV enters that zone, a collision is perceived to have occurred.

The exclusion zone around the UAV can be represented as a circular area within the two-dimensional plane, with the point mass at its centre. The radius of this zone would be double the largest dimension of the UAV, with respect to the centre of mass. The exclusion zone can be described conservatively as

$$R_e = 2R \tag{3.12}$$

where R is the largest dimension of the UAV from its centre of mass, and (R_e) is the exclusion zone radius (R_e). With this exclusion zone in mind, collisions with a UAV at an instant in time can be expressed by

$$\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \leq R_e \quad \forall i, j \in 1, ..., n_r, i \neq j \tag{3.13}$$

where the i^{th} UAV is still represented as a point mass but the UAV with which it may collide (the j^{th} UAV) is represented with an exclusion zone of radius R_e. This zone will be equal for any UAV, due to the assumption that the UAVs are homogeneous.

For the boundary of the environment and the obstacles in the environment, it is easier to include the exclusion zone within their definition. The obstacles are therefore increased in size by a width of R_e and the boundary is decreased in size by the same amount. Assuming this has already been done, a collision of the i^{th} UAV with the environment can simply be expressed as

$$(x_i, y_i) \notin \mathbf{S} \quad \forall i \in 1, ..., n_r \tag{3.14}$$

where $\mathcal{S}$ is the entire search area and being outside of it would cause a collision with obstacles in the environment or with the environment border. Provided Equations 3.13 and 3.14 are not true at any time instant in the search, there will be no collisions.

3.6 Target Model

In the context of SAR, a target is generally a survivor or group of survivors that requires rescue. This may be anyone that is in some form of peril within an environment, and require assistance. For this project, target and survivor will be used interchangeably.

The target may not always by located within the area that is searched by the UAVs, seeing as this search may be part of a larger search plan. For the purposes of this project, it is assumed that the target is within the demarcated area. The case where the target is not in the environment is not accounted for because it would simply result in the UAVs completing their systematic search of the area without locating the survivor(s).

The target is assumed to be static within the environment. This is a valid assumption in the event that someone is trapped in difficult terrain or is injured and cannot move, which is often the case in a rescue operation. Survivors are also often advised to stay put when awaiting a search team.

Given the starting point of a static target within the environment, the location of the survivor(s) can be represented using as

$$X_{\text{g}} = (x_{\text{g}}, y_{\text{g}}) \in \mathbf{S} \tag{3.15}$$

where X_g is the location of the target which is described by the two-dimensional coordinate (x_g, y_g). The entire search area is represented as $\mathcal{S}$. This location would be a hidden goal for the UAVs.

The target is assumed to have a uniform chance of being at any point within this searchable area, even though this region may be part of a larger, probabilistic search plan. Its location is not known to the UAVs, but it is assumed to be a constant for the duration of the search.

The target location is expected to be uncovered by the UAVs during the search. Target detection will be discussed in more detail in Section 3.7. In short, survivor/target detection should be guaranteed if the target is within

range of one of the UAV's onboard sensors. In SAR, false detections (false positives) are more favourable than failed detections [30]. It is assumed that the survivor detection technique would be designed in this fashion and that detection would be guaranteed when in range.

3.7 Target Detection Model

For the purpose of this project it is assumed that the UAVs have onboard cameras. These could be thermal or visual cameras, or both. Historically, these have been used in tandem for survivor/target detection using a UAV. Section 2.2.2.4 discussed an implementation where a thermal camera was used to find potential victims and a visual camera was used to confirm [30].

In general, the consensus for SAR is that a false positive detection is acceptable, but a failed detection is not [30]. This is assumed to be the bias of the algorithm that will be used, and it is therefore assumed that a target will always be detected when in view. In this project a false positive is not explicitly dealt with.

Target detection can be online or offline. The processing can be done onboard the UAV, which may require more processing power, resulting in heavier, more expensive equipment onboard. The processing can also be done in post-processing at a ground station. This requires a stable communication link. Real-time data transfer would be ideal. Section 2.2.2.5 discussed an application where a stable communication link was one of the main goals. It is also possible to implement a hybrid solution. For example, to account for communication failures, UAVs could potentially store enough of the data captured onboard to ensure that the data reaches the ground station once communication is re-established.

Regardless of the survivor detection technique, it is assumed that the target is static and will be detected when within the view of some visual or thermal camera onboard the UAV. A dynamic target would also be detected successfully, if the target is moving slow in relation to the camera FOV. It is important to clarify what it means to be within view. For the case of an onboard camera, this means that the target is detected the moment it enters the field of view (FOV) of the camera. The camera FOV refers to the size of the area on the ground that the camera observes.

The cameras are assumed to be downward facing, so as to enforce a consistent FOV. The roll of the aircraft (ϕ_r) would have an impact on the angle of a camera. However, it is not explicitly considered in this project. It is assumed that there is a gimbal attached to the camera, with an associated control system, that corrects for the roll to ensure the camera continually points downward.

The constant altitude assumption mentioned in Section 3.4 is an advantage when combined with the downward facing camera. It means that the

camera FOV, will remain predictable. This, in turn, means that the maximum ground sampling distance (GSD) can be calculated easily, which is a widely accepted measure of camera accuracy. In this scenario, the GSD is the distance on the ground that corresponds to the width of one pixel in an image [62].

The target has been described as a point in Section 3.6. The target has an overall dimension, but for the purposes of this project, if the point is viewed, it is assumed that the target has been detected. This means that the survivor detection algorithm would need the capability to detect a human even when only partially within the FOV.

Later in this project, the environment will be discretised into a grid using the FOV. However, the grid cell size will be smaller than the actual FOV. In this case the target is considered viewed when the UAV flies through the cell in the grid that the target occupies. There is a chance that detection happens earlier for a real-world application, but this simplification is made for the purpose of simulation.

Chapter 4

System Overview

This chapter provides an overview of the system that was developed to co-operatively search a designated area using multiple unmanned aerial vehicles (UAVs) in support of search and rescue operations. The system is broken down into its various components, and each component is briefly discussed. Section 4.1 summarises the components, how they were implemented, and how they fit together. Sections 4.2 through 4.5 discuss the individual components of the system.

4.1 System Summary and Scope

Figure 4.1 provides an overview of the system and its components. The grey and blue regions highlight the components of the system that are the focus of this project. The purpose of the system is to plan the paths for the UAVs to completely cover the a designated area while avoiding collisions between UAVs and with static terrain, and while obeying the dynamic constraints of the vehicles.

The search region in a SAR operation would be chosen by a search and rescue team. On the diagram, the region to be searched is referred to as the environment. Note that this specifically refers to the area that would be searched by the UAVs. It may however be part of a larger search plan.

The system execution consists of a pre-planning phase and a planning phase. The pre-planning phase consists of environment modelling and environment discretisation. The planning phase consists of the selection of the UAV initial positions, the division of the area into sub-regions, the planning of the individual UAV paths for each sub-region subject to the UAV dynamic constraints, and the flight schedule generation.

The first step is to perform environment modelling of the demarcated search area using the known environment map as an input. The environment modelling includes obstacle identification. Once the environment has been

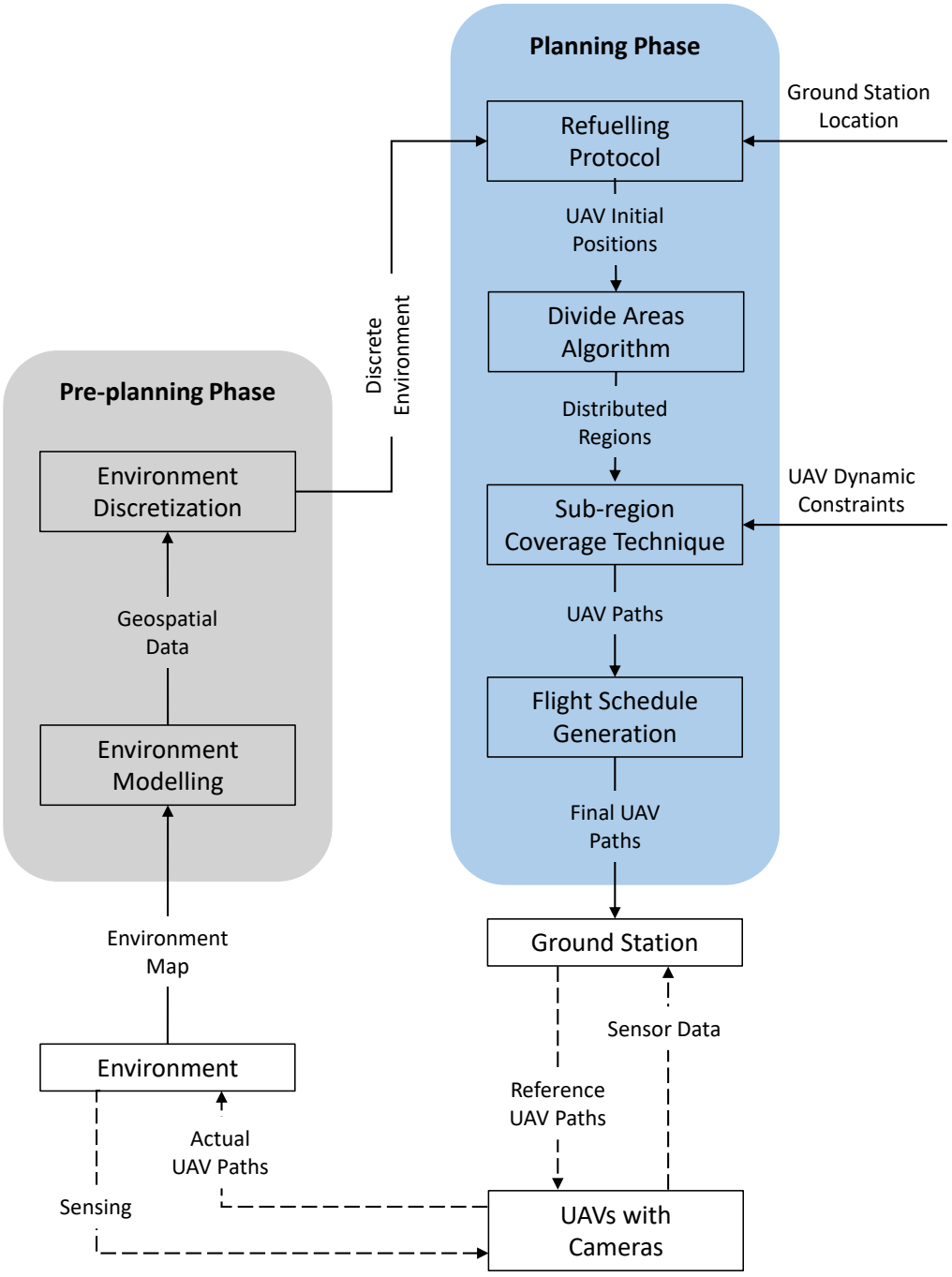


Figure 4.1: Diagram showing an overview of the multi-robot SAR system.

modelled, a series of calculations are performed to choose a suitable search altitude and forward speed for the UAVs.

With this information available, the next step is to discretise the environment into a search grid. This is done using a mathematical process implemented with Python. The discrete obstacles in the environment need to

be identified by the user, based on the actual continuous environment model. The environment modelling and discretisation are collectively called the environment representation.

Assuming there is a central deployment zone for the UAVs, the UAV initial positions can be strategically placed. In this project, the central deployment zone is called the ground station. It is where the UAVs take off and land. Its location in the environment would be chosen by the search team.

If only a limited number of UAVs are available for searching an environment, refuelling or recharging may be necessary. As a whole, the ground station location and the number of refuels necessary would determine the UAV initial positions. The refuelling protocol, which was implemented using Python, will be discussed further in Section 4.5.

The next step in the planning phase of the algorithm is the divide areas algorithm and sub-region coverage technique. Together, these form a distributed and offline coverage path planning algorithm. The algorithm that was used to divide the environment into sub-regions is the divide areas algorithm for optimal multi-robot coverage path planning (DARP). This algorithm seeks to optimally divide an environment among several robots for coverage. The original implementation was done in Java. This original code was used, with some modifications. It was also adjusted to be run from the larger Python script.

The sub-region coverage technique was implemented as part of the larger Python script. The method used is called spanning tree coverage (STC). A bottleneck was found to occur during spanning tree generation, and so this portion of the code, that implements Prim's algorithm, was re-written in Java to reduce the execution time. STC forms closed-loop paths to cover the individual sub-regions. Dynamic constraints were added to these paths to root the problem in a real-world application.

According to Lavalle's book about planning algorithms, everything outside of the shaded regions in the system diagram is called the execution phase [34]. The real-world implementation and testing of these algorithms are beyond the scope of the current project. The system was tested in simulation, and forms the foundation for future work on the practical implementation and testing of the system using physical UAVs.

4.2 Environment Representation

This section describes the components in the grey region of Figure 4.1. This portion of the diagram is called the pre-planning phase. It concerns the process that is followed to convert a three-dimensional, continuous environment into a two-dimensional discrete environment.

The details of this part of the system will be presented in Chapter 5. The main feature of the divide-areas and sub-region coverage algorithms in this system is that they are designed for a rectangular, two-dimensional, grid-based

representation of a search environment. A two-dimensional representation is made possible by using a constant search altitude for the UAVs. A search altitude has to be carefully chosen so that target detection is possible.

The constant flying speed of the UAVs must also be taken into account when discretising the environment, due to the dynamic constraints of the UAVs. The speed should be low enough to allow for 90 degree rotations within one discrete cell, while also adhering to the complete coverage requirement. The search team would be required to make the trade-offs and choose an appropriate search altitude and speed.

Based on the speed and altitude, environment obstacles can then be identified. These could be physical obstructions at the flying altitude. They could also be excluded due to the limitations of the image recognition algorithm that is assumed to be onboard. Certain regions may also be no-fly zones due to air space restrictions.

For the practical examples in this report, the environment modelling and discretisation is done using available topographic maps of real-world locations. These maps are used to identify regions that classify as obstacles and no-fly zones. Based on this continuous, two-dimensional environment model, a discrete representation can be created. The flying altitude and speed determines the discrete element sizes in the environment grid. This element size can then be used to estimate the obstacles in the environment. This concludes the environment representation. The discrete environment representation can now be fed into the planning phase that divides the search area into sub-regions and plans the paths for the individual UAVs to cover their allocated sub-region.

4.3 Divide Areas Algorithm

The divide-areas algorithm takes the initial positions of the UAVs and the grid-based environment as input. The DARP algorithm is used for the area division, and will be discussed in more detail in Chapter 6. The goal of the divide-areas algorithm is to assign sub-regions to the robots in an optimal way.

The divide areas algorithm in Figure 4.1 receives the initial positions of the robots from the refuelling protocol. This assumes that central deployment is used to assign the initial positions, which will be discussed further in Section 4.5.

The main optimization requirement for DARP is to assign sub-regions of approximately equal size to each robot initial position. Essentially, the same number of cells should be assigned to each robot for searching. Paths can then be planned to cover each sub-region. If each sub-region is covered completely, it then follows that the entire environment is covered completely.

The idea is that the UAVs would search these sub-regions in a similar amount of time. Provided they begin their searches simultaneously, they would then finish at roughly the same time, making it an optimized solution.

Aside from equal-sized regions for searching, the DARP algorithm also seeks to form cohesive sub-regions. The UAVs should not be expected to traverse each other's sub-regions to reach their own cells. This also presumes that the initial position of the UAV is within its assigned sub-region.

The DARP algorithm is iterative in nature. Therefore in each iteration, cells assignments are exchanged between UAVs until it converges to a solution that meets the requirements of a near-optimal solution. If contiguous sub-regions can be formed, it means that UAV collisions should not occur during flight. Not needing additional collision avoidance algorithms is a significant advantage in a multiple robot application.

Note that some modifications were introduced into the original algorithm. The most prominent change is that the option was added to change the distance measure used when dividing the regions between robots.

4.4 Sub-Region Coverage Technique

The sub-regions produced by the divide areas algorithm must be searched by the individual UAVs assigned to them. There will be one UAV initial position associated with each sub-region. Note in Figure 4.1 how the sub-region coverage algorithm takes the distributed regions from the divide areas algorithm as input.

This project makes use of spanning tree coverage (STC) to cover the sub-regions. The specific algorithm that is used is a minimum spanning tree (MST) algorithm referred to as Prim's algorithm [63]. A contiguous sub-region grid is used to create a graph that is fed into this algorithm. The resulting spanning tree can then be circumnavigated to achieve coverage of the sub-region. The path is a closed loop, meaning that the final and initial positions in the flight path are the same.

Since it is a MST, weights at the edges of the graph can be used to favour a certain search direction. In this implementation of the algorithm, dynamic constraints are also added to each turn manoeuvre.

The paths generated are expected to be executed at a constant speed. They also have an associated path length. This distance and speed can be used to calculate an associated time. The time calculated can be adjusted to represent energy consumption during flight. A flight is only feasible if it is within the energy constraints of the UAV. Chapter 7 will discuss the sub-region coverage technique in more detail, along with the approach that was used to modify the paths to accommodate the dynamic constraints of the UAVs.

4.5 Central Deployment and Scheduling

This section discusses two components of the diagram in Figure 4.1. The refuelling protocol is used to calculate the number of sub-regions that should be created by the divide-areas algorithm. This in-turn produces a number of robot initial positions, which are a multiple of the available UAVs. This takes the endurance limitations of the UAVs into account. The flight schedule generation generates the flight plans that allow the UAVs to take off and land sequentially at the central ground station.

The refuelling protocol takes the ground station location as input. The location of this would be chosen by the search team. This is from where the UAVs are launched, as well as where they land. The UAVs can start their approach for landing and end there departure after take-off at the same location which are the initial positions of the UAVs in their closed-loop sub-region coverage paths.

The number of UAVs, when refuelling is considered, would not necessarily be equivalent to the number of sub-regions. A number of sub-regions could now be assigned to the same UAV, and it would simply search them sequentially. Due to endurance limitations, the UAVs would refuel between the sequential sub-region searches assigned to them. The refuelling protocol in this report is also set up so that the available UAVs always refuel the same number of times.

Landing and take-off are considered to be the manoeuvres to move to and from the search altitude. Approach and departure, in turn, refer to manoeuvres that take the UAVs between the ground station and their initial positions. The UAV initial positions are arranged around the ground station to make these approach and departure manoeuvres relatively short. Therefore a larger portion of the endurance capacity of the UAV is used for the actual search.

Since this is an offline planner, the number of refuels are calculated prior to knowing the actual flight path lengths. The number of UAV initial positions are directly correlated with the number of refuels, so a method was devised to estimate the number of refuels that would be necessary. The initial positions of the UAVs can then be arranged around the ground station to be fed into the divide areas algorithm. The final paths and corresponding flight schedule can then be generated.

The full time taken and energy consumed can then be calculated and compared to the original energy constraint. If the flight plan is within the energy constraints of the UAV being used, the final paths are feasible and the algorithm has successfully found a solution. If not, a rerun of the algorithm would be necessary with more sub-regions assigned per UAV.

This flight plan can be used for a real world system that uses offline planning. Survivor detection would occur during flight. The location of the survivor is not known a priori. However, in this project a number of simulated survivor locations are used to test the algorithm. Survivor detection times can therefore be calculated. A snapshot of the environment at the moment of

detection can also be shown for any simulated survivor location.

The refuelling protocol and central deployment mechanism form the basis for applying the coverage path planning algorithms in real-world SAR operations. Accounting for endurance limitations make this implementation feasible for use with UAVs that have a finite amount of fuel, while a flight schedule eliminates collisions at the ground station. The divide-areas algorithm eliminates any further collisions while the UAVs are executing their coverage paths, and dynamic constraints of the UAV are accounted for in the individual subregion coverage technique. All these mechanisms combined result in a feasible path planning technique for use in practice.

Chapter 5

Environment Representation

Section 5.1 discusses some background regarding the modelling of environments. Section 5.2 then discusses in detail how discretisation of the continuous environment will be executed. Section 5.3 then briefly covers the different unmanned aerial vehicle (UAV) and camera options available before Section 5.4 shows examples of the actual discretisation process executed on four different real-world scenarios.

5.1 Background

Assuming that an area for searching has already been demarcated for the UAVs, the next challenge would be to model the environment. Although the final representation of the environment will be a two-dimensional one, the original model would need to be three-dimensional.

From the three-dimensional model, one can ascertain which obstacles are at an altitude that would obstruct the UAVs. These would be obstacles that extend to the constant altitude at which the UAVs fly and beyond. A description of the types of obstacles and how they are described within the context of an environment was given in Sections 3.2 and 3.3.

One way to describe a three-dimensional environment is a digital elevation model (DEM), which is a type of geographic information system (GIS) layer. DEMs can generally be divided into digital surface models (DSMs) and digital terrain models (DTMs). A DSM is a three-dimensional rendering of the Earth's surface with all the objects present on it. These objects, whether man-made or natural, may become obstacles for UAVs in flight. A DTM is generally a representation of the bare earth without any objects present on it. [64]

There are a number of ways in which to generate a DEM. Ground surveying is an option using a theodolite or a differential GPS (DGPS). These methods generate discrete data that need to be interpolated to create a DSM in the form of a continuous raster. Both methods also require skilled labour and for terrain that is difficult to traverse, these methods become impracti-

cal. Another option is to digitise and interpolate contour lines from existing contour maps, but this is also rather labour intensive and generates a DTM rather than a DSM. [64]

Data is often collected using sensors aboard some type of aircraft. synthetic aperture radar interferometry (inSAR) is a method that creates a DEM using simultaneously captured radar images from multiple different antennas [64]. This data is gathered using a space shuttle, and the latest release by NASA in 2014 boasts a resolution of roughly 30 metres [65].

Another example is photogrammetry, wherein images are taken of the same area from multiple points in order to yield perspective, similar to how the human eye perceives depth. Lastly, there is a method called light detection and ranging (LiDAR), wherein light is measured as it reflects off the Earth's surface in order to formulate a DSM. This generates large point clouds of high accuracy data, but is generally quite expensive to implement. [64]

A DEM may be available for certain regions. If a recent DEM for a particular region is available, it would be useful to use this as a method of identifying obstacles in a demarcated area. It would be particularly advantageous if an accurate DEM, such as a LiDAR scan is available. However, there are other possibilities. For example, in a mountainous region, a contour map may suffice. In other types of environments it may only be necessary to know which areas are no-fly zones, seen as there would not be obstructions tall enough to come into the path of the UAVs.

The altitude at which the UAVs fly would be a big factor in determining the obstructions and no-fly zones. There may also be areas that would not get searched using a UAV due to a lack of map data.

5.2 Discretisation Methodology

In this section a proposed environment discretisation technique is formulated, taking into account the required ground sampling distance (GSD) for viable target detection. The requirement of complete coverage is also taken into account, which is found to rely on the dynamic constraints of the UAV at a constant speed.

5.2.1 Target Detection Requirement

Seeing as target detection is the main objective for the UAVs in a search and rescue (SAR) operation, it becomes a limiting factor in designing discretisations for a continuous environment. With the assumption that the UAVs are flying at a constant altitude and that they have downward-facing cameras, a field of view (FOV) can be calculated. This is the size of the area on the ground that the camera observes. Target detection occurs when the target is within the camera FOV.

FOV is camera lens dependant. The diagram in Figure 5.1 shows all the relevant variables needed to calculate the cross-track field of view (FOV_x). A similar diagram can be used to calculate the in-track field of view (FOV_y). The only difference would be the sensor size variable, which changes from the sensor width (w_{len}) to the sensor height (h_{len}). The other variables include the focal length of the camera (f), the height of the lens above ground (H) and the camera's angle of view (AOV). Lastly, there is the variable α which is an angle created due to the sensor being slightly smaller than the diameter of the cone of light projected onto it. The resulting FOV will be a rectangle of the same aspect ratio as the camera sensor, provided the ground is level.

The assumption that the ground is level is not an accurate one, since most environments have topographical variations. However, if the GSD calculation is done for the lowest point in the terrain, any variations in the terrain would only result in lower, more favourable GSD values. Assuming level ground is therefore a conservative approach. Similarly, if the FOV is calculated for the highest topographical point in an environment, the FOV for any point below this would simply result in larger regions being covered by the camera.

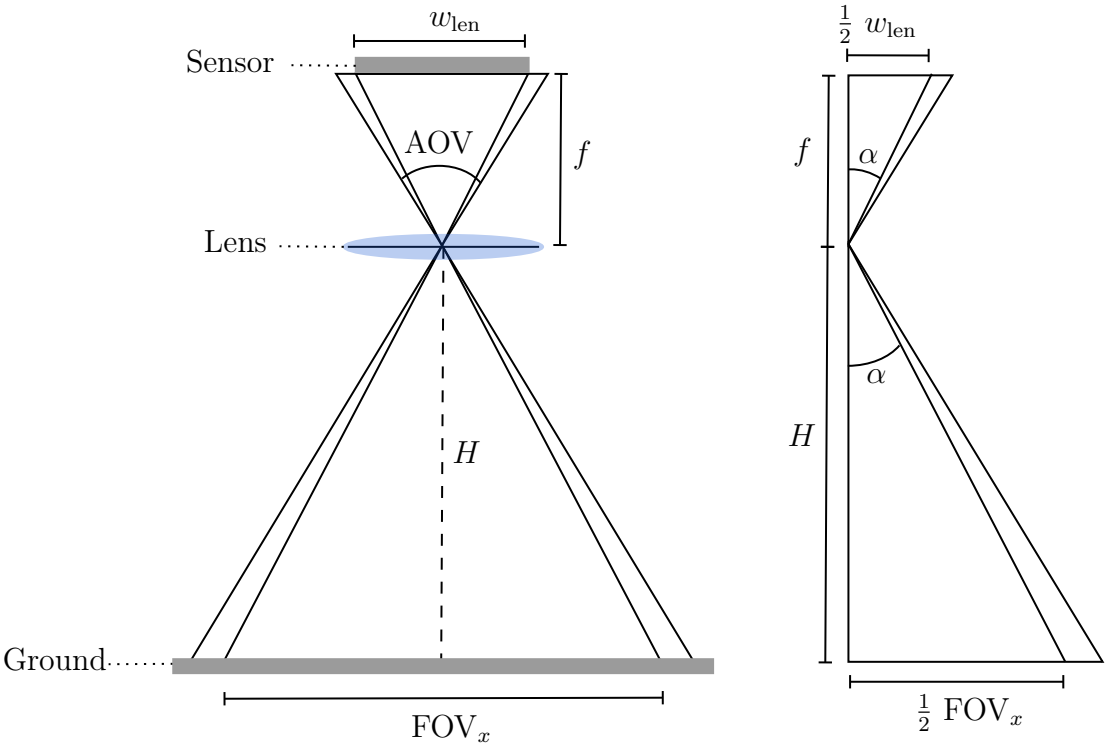


Figure 5.1: Diagram showing relevant variables concerned with calculating the field of view for a camera.

To calculate the FOV, the first equation that is required is the calculation of the angle α, which makes use of the small triangle in Figure 5.1. The

equation is expressed as

$$\begin{aligned} \tan\alpha &= \frac{\frac{1}{2}w_{\text{len}}}{f} \\ \alpha &= \tan^{-1}\left(\frac{w_{\text{len}}}{2f}\right) \end{aligned} \tag{5.1}$$

where w_{len} is the sensor width and f is the focal length of the onboard camera. Now that α is known, the larger triangle is used to calculate FOV_x with

$$\begin{aligned} \frac{\text{FOV}_x}{2} &= H \cdot \tan\alpha \\ \text{FOV}_x &= 2H \cdot \tan\left(\tan^{-1}\left(\frac{w_{\text{len}}}{2f}\right)\right) \\ \text{FOV}_x &= H \cdot \frac{w_{\text{len}}}{f} \end{aligned} \tag{5.2}$$

where H is the height of the camera sensor above ground and FOV_x is the cross-track camera field of view for level ground. The other dimension of FOV can be calculated similarly with

$$\text{FOV}_y = H \cdot \frac{h_{\text{len}}}{f} \tag{5.3}$$

where h_{len} is the other camera sensor dimension and FOV_y is the in-track camera field of view. It should be noted that FOV_x and FOV_y will have the same units as H, which is metres.

The resolution of the camera is the number of pixels along the image width multiplied with the number of pixels along its height. These can be used to calculate the ground sampling distance as follows,

$$\begin{aligned} \text{GSD} &= \frac{100\ \text{FOV}_x}{\text{px}_w} \\ \text{GSD} &= \frac{100\ H \cdot w_{\text{len}}}{f \cdot \text{px}_w} \end{aligned} \tag{5.4}$$

where px_h and px_w are the two pixel values, and FOV_y and FOV_x are the associated field of view values. The FOV value is multiplied by 100 to get the value in centimetres instead of metres, since GSD is conventionally shown with a unit of cm/px.

GSD can be used as a constraint when choosing the altitude at which the UAVs must fly. One would want the UAVs to fly low enough so that the GSD is still accurate enough to detect the target.

The calculations for FOV are dependant on camera altitude and certain parameters specific to the camera used. The GSD is in turn dependant on both the FOV and camera resolution. GSD is therefore both camera- and altitude-dependant.

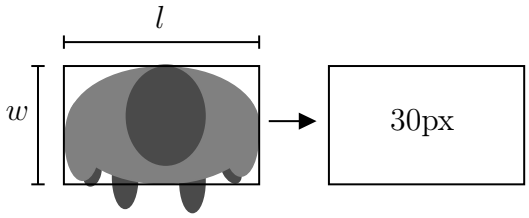


Figure 5.2: Diagram showing the rectangular approximation for a human viewed from above for calculation of the GSD

If the size of the environment discretisation is set equal to the rectangular camera field of view, and the type of camera used is known, then a desired GSD can be used to decide on an appropriate flying height. Choosing a GSD would depend on the application, seeing as it represents the level of detail that can potentially be detected in an image.

Taking the conservative approach, the assumption is that one is looking for a human standing upright, viewed from above. To get a good estimate of the space occupied by a human in this orientation, one needs anthropometric data. A survey was done in Europe for people between the ages of 18 and 60 years [66]. Among other measurements, they measured chest depth (w) and elbow-to-elbow length (l). These dimensions represent those of an upright human from above and as shown in Figure 5.2.

To calculate GSD, a minimum number of pixels needed to make a human visible to an image recognition algorithm must be chosen. Rudol and Doherty [30] developed an image processing algorithm for human detection in a search and rescue scenario . In their paper they made the decision to put a 30 pixel requirement on human detection. Figure 5.2 shows the rectangular approximation for a human that the 30 pixels should represent.

Using the lower percentile measurements of 170mm chest depth and 390 mm elbow-to-elbow length along with the 30 pixel requirement, one gets a GSD of roughly 4.7cm/px. This value will be used to calculate appropriate flying heights, which is a reasonable value considering that most aerial surveys operate at a GSD of less than 5cm/px [62]. This value would be dependent on the method of target detection, but this estimation will be used for the purpose of this project.

The calculation for the maximum allowable height using a GSD requirement is expressed as

$$H_{\max} = \frac{\mathrm{GSD}_{\max} \cdot f \cdot \mathrm{px}_w}{100 w_{\mathrm{len}}} \tag{5.5}$$

where $\mathrm{GSD}_{\max}$ is the maximum allowable GSD to guarantee target detection, f is the camera's focal length, px_w is the number of pixels along the image width for this camera, and w_{len} is the camera sensor width. A factor of 100 is used to convert the GSD from a unit of cm/px to a unit of m/px, so that the resulting maximum flying height ($H_{\max}$) is in metres.

This height would be with respect to the lowest point in the topography of the environment. This ensures that any topographical variation would simply result in a lower GSD, and therefore higher accuracy.

When choosing a flying altitude for the search, one equal to or lower than $H_{\max}$ should be chosen. Choosing higher altitudes means a larger FOV which leads to faster coverage. However, a search team may choose to favour having a lower GSD and may therefore prefer lower altitude searches. Since the proposed method of environment discretisation is based on the FOV, it means that the obstacle resolution would also be higher at lower altitudes.

5.2.2 Complete Coverage Requirement

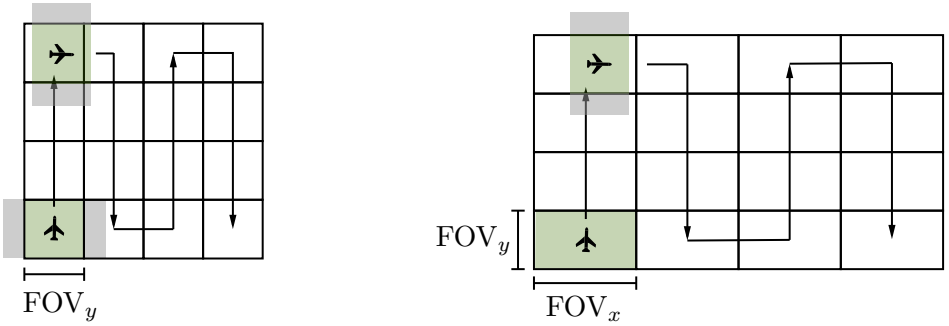
Target detection is only reliable when complete coverage is achieved. If portions of the map that the UAVs are expected to cover are in fact not covered, it is possible to miss a target entirely. In this section it will be shown that the dynamic constraints of the UAVs impose limitations on the height at which the UAVs must fly above the ground in order to still providing complete coverage.

The FOV can be used as a guideline when deciding how to discretise the environment. A FOV is generally rectangular, most commonly with an aspect ratio of either 4:3 or 3:2. Assuming that a UAV will execute either a straight line manoeuvre or a circular one within a single discretised cell in the grid, one needs to ensure that complete coverage is still achieved. Once the UAV enters a discrete cell, that cell should be completely covered before the UAV exits it again.

A good number of single robot area coverage techniques employ something resembling a sweep. Several of these techniques were discussed in Section 2.5. The technique that was chosen for this project is spanning tree coverage (STC). This will discussed further in Chapter 7. The important thing to note with this technique is that within a discrete cell, the UAV is either expected to move in a straight line or turn through 90 degrees or turn through 90 degrees using a semi-circular maneuver.

The discretisation for this implementation is assumed to be either rectangular or square to correspond to the FOV shape. If it is assumed that the UAV is omnidirectional, choosing these discretisation sizes would be simplified greatly. For the square scenario, one can simply set the sides of the square equal to FOV_y to ensure complete coverage. An example of this discretisation is shown in Figure 5.3a. For the rectangular scenario, the discretisation can be set exactly equal to the size of the FOV rectangle. This is illustrated in Figure 5.3b.

The square scenario would have cross-track overlap when executing movements in either dimension. The rectangular discretisation would only have cross-track overlap in one dimension. In both figures, redundant coverage as a result of this overlap is shown in grey. An environment would be covered faster



(a) Square discretisation (b) Rectangular discretisation

Figure 5.3: Diagrams showing cross track overlap for camera Field of View for different discretisation techniques.

with the rectangular discretisation provided the direction with no cross-track overlap is favoured during flight.

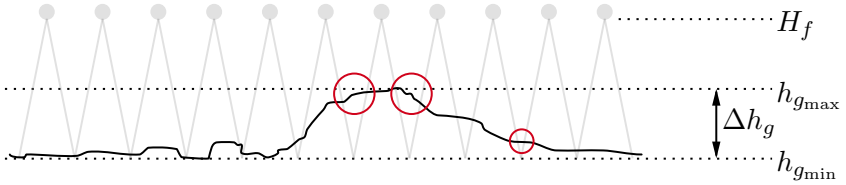
The height used in the calculation of this FOV would be with respect to the highest point in the topography of the environment. This ensures that any topographical variation would simply cause the FOV size to increase, leading to more overlap. Figure 5.4a shows what may happen if the lowest point in the topography is used as reference. The red circles show regions that are not being covered. Figure 5.4b then shows how complete coverage is achieved when using the highest point as a reference.

This technique inevitably leads to smaller discretisation, and therefore more time-consuming coverage. Guaranteed target detection is a priority for SAR, though. Complete coverage with more overlap is therefore favoured over faster, and possibly incomplete, coverage.

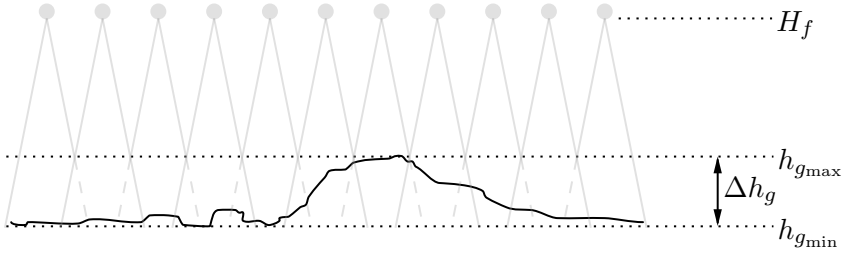
When considering the dynamic constraints of the UAV, sharp 90 degree turns are no longer possible. Because this application seeks to implement constant speed searches, even multi-rotors would not be considered omnidirectional. A 90 degree turn, where a limited turning radius is assumed, is shown in Figure 5.5.

When creating a discretisation with this in mind, the first constraint is that the UAV must complete a 90 degree turn within one discrete cell. The second constraint is that the discrete cell must be fully covered by the camera during this manoeuvre. With the first constraint, the UAV minimum turning radius needs to be considered at the chosen constant speed. This calculation is shown in Equation 3.7. The second constraint demands that as the UAV rotates, it needs to cover the entire discrete cell. This calculation is complicated by using a rectangular discretisation, so a square discretisation is assumed. To achieve this, the FOV needs to reach the furthest corner of the discrete cell when at the apex of its turn.

Figure 5.6 shows the dimension (L) that needs to be equal to half of the cross-track FOV. The calculation for this dimension, using relecant dimensions



(a) H_f with respect to lowest point in topography



(b) H_f with respect to highest point in topography

Figure 5.4: Diagrams showing coverage achieved when FOV is calculated using a height (H_f) with respect to different points in topography.

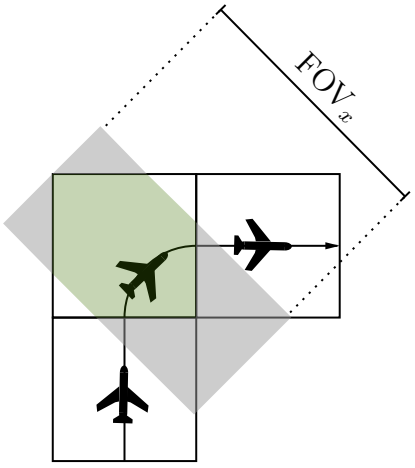


Figure 5.5: Diagram showing required FOV to ensure no corner cutting on a square. discretisation

portrayed in the figure, is as follows

$$\begin{aligned} L &= 2\sqrt{2}\, r_{\min} - r_{\min} \\ \frac{\mathrm{FOV}_x}{2} &= 2\sqrt{2}\, r_{\min} - r_{\min} \\ \mathrm{FOV}_x &= r_{\min}(4\sqrt{2} - 2) \end{aligned} \tag{5.6}$$

where FOV_x is the cross-track camera FOV and $r_{\min}$ is the minimum UAV turning radius at the constant forward. Using the FOV, one can also calculate

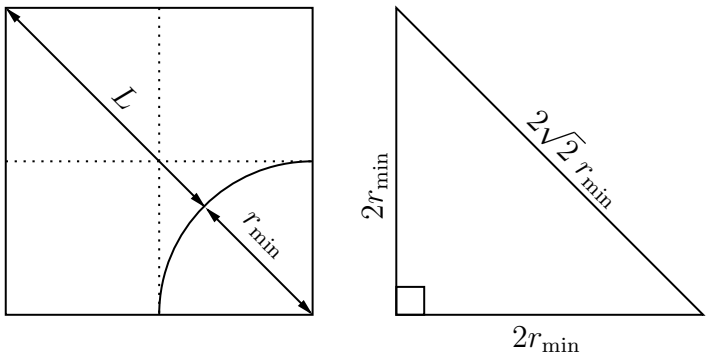


Figure 5.6: Diagram showing the dimensions necessary to calculate FOV on a square discretisation.

the minimum height that the UAV can fly at this speed using

$$H_{\min} = \frac{\text{FOV}_x \cdot f}{w_{\text{len}}} \tag{5.7}$$

where $H_{\min}$ is the is the minimum height of the UAV, above the highest point in the topography, to guarantee complete coverage. This highest point is labelled $h_{g_{\max}}$ in Figure 5.4b. Based on the chosen GSD of 4.7cm/px, Equation 5.5 calculates a maximum allowable height at which the UAV should fly. This height would be from the lowest point in the topography, labelled $h_{g_{\min}}$ in Figure 5.4a.

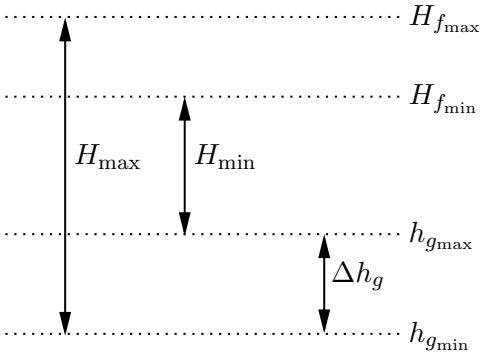


Figure 5.7: Diagram showing the range of flying heights that can be calculated.

Figure 5.7 shows how these heights can be used to calculate a range of possible flying heights at this speed. The environment discretisation can only be performed if the variation in the ground height is less than or equal to the difference between the minimum height required for coverage and the maximum height allowed for detection. This constraint is expressed mathematically by

$$\Delta h_g \leq H_{\max} - H_{\min} \tag{5.8}$$

where Δh_g is the difference between the highest and the lowest point in the search area, $H_{\min}$ is the minimum height above ground that the UAV must fly to provide complete coverage, and $H_{\max}$ is the maximum height above the ground at which the the onboard camera can still reliably detect the target.

The allowable flying altitude range for the UAV is therefore given by

$$H_{f_{\min}} \leq H_f \leq H_{f_{\max}} \tag{5.9}$$

where H_f is the actual UAV flying altitude and $H_{f_{\min}}$ and $H_{f_{\max}}$ are the minimum and maximum flying altitudes allowed. The calculation for these minimum and maximum flying altitudes is given by

$$\begin{aligned} H_{f_{\min}} &= h_{g_{\max}} + H_{\min} \\ H_{f_{\max}} &= h_{g_{\min}} + H_{\max} \end{aligned} \tag{5.10}$$

where $h_{g_{\min}}$ and $h_{g_{\max}}$ are the altitudes of the lowest and highest point in the topography respectively. This excludes obstacles and is strictly for the area that needs to be searched. $H_{\min}$ is the minimum flying height of the UAV with respect to $h_{g_{\max}}$, and $H_{\max}$ is the maximum flying height of the UAV with respect to $h_{g_{\min}}$.

The minimum flying altitude is therefore the maximum ground altitude plus the minimum flying height above the ground that is required for complete coverage. The maximum flying altitude is the minimum ground altitude plus the maximum height above ground that is required for target detection.

Flying higher would be favourable for faster coverage, but flying lower would allow for a smaller obstacle resolution.

It is useful to note that the requirement for the range of heights can be used as a tool in choosing one's search environment boundaries and obstacles. With the assumption that the topography variation is zero ($\Delta h_g = 0$), one can calculate $H_{\min}$ and $H_{\max}$. The difference between them would then work as a maximum allowable topographical variation for complete coverage and target detection to be achievable. This is assuming the flying speed is a known value.

A suggested maximum speed can be calculated for the UAV, which would help ensure that $H_{f_{\min}}$ is less than $H_{f_{\max}}$. One can rework Equation 5.6 to calculate a suggested minimum turning radius for the UAV at the maximum height as follows,

$$\begin{aligned} r_{\min} &= \frac{\text{FOV}_x}{4\sqrt{2}-2} \\ r_{\min} &= \frac{H_{\max} - \Delta h_g}{4\sqrt{2}-2} \cdot \left(\frac{w_{\text{len}}}{f}\right) \end{aligned} \tag{5.11}$$

where $r_{\min}$ is the suggested minimum turning radius and fov_x is the cross-track camera FOV, calculated using the maximum height of the UAV above the highest point in the topography. H_max is the maximum height from the lowest point in the topography and Δh_g is the topographic variation in the search

area. Equation 3.7 can then be reworked to calculate a suggested maximum constant speed at which the UAV should fly for this minimum turning radius as follows,

$$V_{\max} = \sqrt{r_{\min} \cdot g \tan(\phi_{\max})} \tag{5.12}$$

where $V_{\max}$ is the suggested maximum speed, $\phi_{\max}$ is the maximum bank angle of the UAV, and g is the gravitational acceleration constant. A speed lower than this maximum, but higher than the stall speed of the UAV can then be selected. The ideal speed would be the cruise speed of the UAV, since this is where it flies the most efficiently. The range of possible speeds are expressed as

$$V_{\text{stall}} \leq V_f \leq V_{\max} \tag{5.13}$$

where V_f is the constant forward speed for the UAV, V_{stall} is the stall speed, and $V_{\max}$ is the suggested maximum speed that corresponds to the minimum turning radius $r_{\min}$ of the UAV.

5.2.3 Discretisation Technique

Once the speed (V_f) and height (H_f) for the search operation are chosen, it is time to calculate the final size of the square discretisation. This is also assuming a camera has been selected and its relevant parameters are known. The overall process is described in the method below:

1. Use the chosen height to calculate the actual camera FOV with Equations 5.2 and 5.3. Note that this should be relative to the highest point within the environment ($h_{g_{\max}}$).

2. Use the chosen speed to calculate the minimum turning radius of the UAV with Equation 3.7.

3. Conservatively assume that the turning radius of the UAV will be equal to half the dimension of the square discretisation. Now, Equation 5.14 can be used to calculate the size of this square discretisation using

$$l = \frac{\text{FOV}_x}{2\sqrt{2} - 1} \tag{5.14}$$

   where l is the dimension of the square discretisation and fov_x is the cross-track FOV

4. Check that the the calculated minimum turning radius is in fact less than or equal to half the dimension of the square discretisation. If it is, then this is an appropriate discretisation size for this combination of speed and height. Note how this generates a range of acceptable turning radius values for this descretisation size, with $r_{\min}$ as the lower limit and $\frac{l}{2}$ as the upper limit. For the purpose of this project however, the aircraft

is assumed to be flying at the minimum turning radius. The condition used is expressed as

$$r_{\min} \leq \frac{l}{2} \tag{5.15}$$

5.2.4 Cross-track overlap

Once a flying height and speed is selected, it is useful to have a metric that represents the amount of redundant coverage. In this case, it is most useful to calculate the cross-track overlap. For the purpose of this project, this would be the overlap of FOV_x as the UAV moves in a straight line over a square discretisation of dimension l. The calculation is shown in Equation 5.16.

Figure 5.3a shows the cross-track overlap for a square discretisation, when the UAV is moving in a straight line. They green represents the portion of the FOV that covers the discrete cell, whereas the grey portion represents the amount of cross-track overlap in that instance.

It is important to note that FOV_y may not be equal to the square dimension (l) as in that example. Because this is however the direction in which the UAV will be moving, coverage should be guaranteed regardless. In-track overlap is a less meaningful measure since there is continuous movement along that direction. Only the cross-track percentage overlap is therefore defined, as follows

$$\%\text{Overlap} = \frac{\text{FOV}_x - l}{l} \cdot 100 \tag{5.16}$$

where FOV_x is the cross-track FOV and l is the dimension of the square democratisation used to form a gird-based environment. This overlap percentage is simply a value to represent relative redundant coverage and should not be accepted as an accurate measure.

There is an additional overlap that occurs as a result of topographic variations. The camera FOV can be calculated with respect to the lowest point in the topography to consider the worst case scenario overlap. Similarly, one could use the highest point in the topography to calculate a best case scenario.

Note that this overlap may cause early target detection in practice. This is not considered for the sake of this project. It is assumed that the target is detected when the UAV traverses the discrete cell in which the target is present.

5.3 UAV and Camera Payload

There are many options to choose from for UAV and camera combinations. The goal of this project is not to do an in depth study of the different options available, however there are a few key factors to note.

With UAVs there are multi-rotor, single-rotor, and fixed-wing options. Fixed-wing aircraft have longer flight times and higher payload capacities. Longer flight times allow for larger areas to be searched and higher payload capacities allow better cameras to be mounted. Both of these are favourable for survivor detection.

Within this class of aircraft there are a myriad of options. Their designs vary to accommodate applications in surveillance, mapping, aerial photography, military defense, firefighting, structure inspections, and more.

They may vary in engine type, overall size, payload capacity, cruise speeds, ability for vertical take off and landing (VTOL), endurance, and resistance to environmental factors such as wind, rain, and snow.

There are also several different camera options that can be mounted to the UAV. Based on the camera parameters, a range of possible UAV heights and velocities can be determined during discretisation of an environment. The camera is considered the payload of a UAV and so the UAV must be equipped to handle this payload.

The calculated range of velocities and heights need to be within the UAV capabilities. Therefore it is important to choose an appropriate UAV and camera combination for a specific environment.

Table 5.1 shows several different options for UAVs and Table 5.2 shows a number of possible camera payloads. These are by no means an exhaustive list. They simply illustrate the range of products that are available, and some of them will be chosen for the scenarios in Section 5.4.

UAV Name	Cruise Speed (m/s)	Speed Limits (m/s)	Payload Capacity (kg)	Weight (kg)	Altitude Limits (m)	Flight Time	Take-off	Power Source	Intended Application	Ref
Wingtra I	16	-	0.8	3.7	<5000	42-59min	VTOL	Li-ion Battery	Surveying and Mapping	[67]
GULL 24	<38	-	18	-	-	-	STOL	Piston Engine	Naval and Coastguard Applications	[68]
Albatross	19	<35	4.4	10	-	<4hrs	Runway	Li-ion Battery	Open Source Long Range Applications	[69]
Sea Cavalry SD-40	25-40	<50	6	34	<5000	<6hrs	VTOL	Battery	Maritime Patrol, Emergency Rescue	[70]
AVEM	18	-	0.5	2	<3500	<3hrs	Hand Launch	Battery	Aerial Mapping for GIS	[71]
D-Sentry	30	>20	1	7.5	<4000	<1hrs	VTOL	Battery	Aerial Mapping, Surveillance, Search and Rescue	[72]
Strix 400	14-25	7-33	2.5	6.5	<5000	<10hrs	Hand Launch	Li-ion Battery and Solar	Surveillance, Endurance Applications	[73]

Table 5.1: List of possible UAVs with their associated capabilities and limitations.

Camera Name	Camera Type	Focal Length (mm)	Sensor Type	Sensor Size (mm)	Resolution	Payload Weight (g)	Maximum Height (m)	Commonly Paired with UAVs	Ref
Sony RX1R II	RGB	35	Full frame	35.9 x 24	8000 x 5320	590	367	Wingtra I, Trinity F90+	[67]
Sony a6100	RGB	20	APS-C	23.5 x 15.6	6000 x 4000	550	240	Wingtra I Albatross	[67]
MicaSense RedEdge-MX	RGB, RE, NIR	5.5	Five Sensors	4.8 x 3.6	1280 x 960	380	69	Wingtra I	[67]
MicaSense RedEdge-P	RGB, RE, NIR	5.5	Five Sensors	5.04 x 3.78	1459 x 1088	502	75	Wingtra I	[67]
Flir Vue Pro	Thermal IR	13	Uncooled VOx Microbolometer	2.88 x 2.05	336 x 256	72	71	Albatross	[74]
Mavic Air 2 Camera	RGB	4.4	1/2" CMOS	6.4 x 4.8	4000 x 3000	-	129	Mavic Air 2	[75]

Table 5.2: List of possible camera payloads with their associated parameters and limitations.

The maximum height (H_{max}) value is included in the list of possible cameras. For this calculation a GSD of 4.7cm/px is used as the limiting value, as calculated in 5.2.1. From this it is clear to see why the Sony RX1R II is one of the most popular cameras in use. It outperforms the rest of the options quite significantly, and so will be used for all the scenarios presented in the following Section 5.4.

The UAVs used will however vary. For both mountainous examples, the Strix 400 will be used. The ground SAR example will make use of the Wingtra I, and the marine example will make use of the GULL 24. These are intuitive choices that will be elaborated on in the respective sections.

5.4 Discretisation Examples

This section shows examples of the proposed discretisation methodology being applied to four different scenarios in the context of South Africa. In each scenario it is assumed that the proper permissions were acquired to fly over private property, roads, and other restricted regions. It is also assumed that flying at these altitudes is prohibited and that no manned aircraft will enter this airspace for the duration of the search.

A fair amount of technical knowledge would be necessary to do these discretisations by hand for a real-world SAR operation. It is likely that a form of software would need to be developed to automate this process to a degree, in order for it to be more practical for use in a real-world application.

The topographic maps used for the examples in this section were obtained from a South African topographic maps website [8]. The corresponding satellite images were found using either Google maps [7] or Mapcarta [9].

5.4.1 Low Altitude Mountainous SAR

For mountainous terrains, the topography may vary significantly. The availability of a contour map is therefore essential to ascertain where the UAV can reasonably fly.

Environments like these are challenging for image recognition software and make target detection a challenge. It is therefore important to carefully consider possible flying heights with respect to the variations in topography.

This example is of a region north of Caledon in the Western Cape. The central mountain in this area is Spitskop, which has a peak height above sea level of roughly 608 m. A possible scenario in this case is hikers who have gotten lost.

Figure 5.8 shows a satellite image of the area that is intended to be searched. Figure 5.9 shows the associated topographical map which serves as the starting map for this section. The real world size of these maps is about 15.3 km by 7.7 km.

The UAV chosen for this application is the Strix 400. It has long flying times which may be necessary for the UAVs to move between the launch zone and the search area. Due to the mountainous terrain, taking off and landing in the search area may be a challenge. Furthermore, it has an appropriate payload capacity for mounting a Sony RX1R II camera. It also has a lower cruise speed, which proves useful for applications with large topographic variations.

With the starting point of the Sony RX1R II, $H_{\max}$ can be calculated using Equation 5.5. The resulting height, that corresponds to a GSD of 4.7 cm/px, is 367 m.

One can now calculate a suggested maximum speed for the UAV using this maximum height as the flying height. To simplify the calculation, it is initially assumed that there is no topographic variation ($\Delta h_g = 0$). Using this assumption with Equations 5.11 and 5.12, one gets a suggested maximum speed of 21.7 m/s.

Since the 14 m/s cruise speed of the Strix 400 is substantially lower than this, it can be used as the constant flying speed for this example. This speed can be used to calculate a minimum allowable flying height ($H_{\min}$) for the UAV. Using Equation 5.7, this value turns out to be 153 m.

Since the minimum height is measured from the lowest point in the topography, and the maximum height is measured from the highest point in the topography, one can use these values to calculate a maximum allowable topographic variation for the environment ($(\Delta h_g)_{\max}$).

This is done using the requirement shown in Equation 5.8, which is reworked as shown in Equation 5.17. The resulting maximum variation becomes 214 m.

$$(\Delta h_g)_{\max} = H_{\max} - H_{\min} \tag{5.17}$$

Figure 5.8: Satellite image of Spitskop example environment. [7]

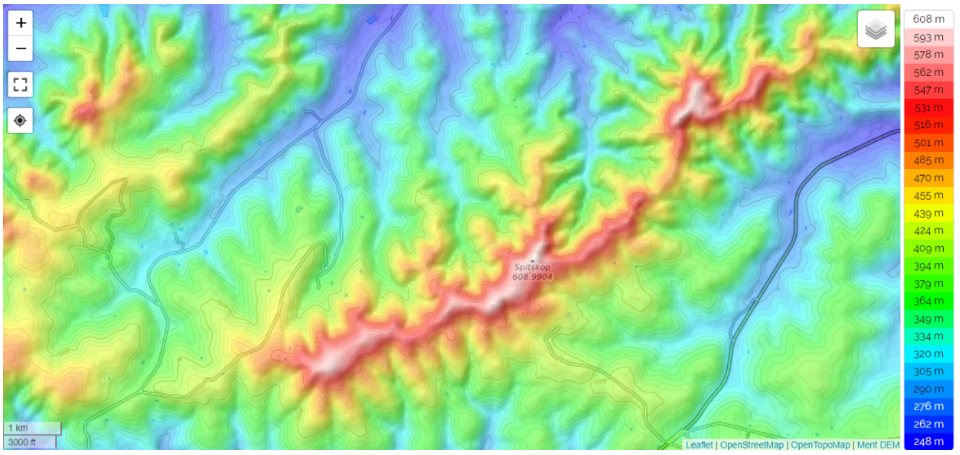


Figure 5.9: Contour map of Spitskop example environment. [8]

The environment for this example is shown in Figure 5.9, which has an overall topographic variation of 360m. From this it is clear that the entire region can not be searched with this UAV and camera combination.

When looking at Equation 5.10, it is clear that the range of allowable flying heights depends on the topographic variation. Therefore, one can choose a topographic variation as a means to choose a flying height. It is best to choose one lower than the maximum allowable variation, and so a value of 200 m is chosen.

The search team can then decide how best this 200 m variation can be utilised. For this application it is assumed that searching between the altitudes above sea level of 300 m and 500 m would be the most useful.

The reason for this would most likely be due to predictions for plausible survivor location. These heights are also chosen partially due to the availability of contour lines at these altitudes.

With these height calculations, the diagram in Figure 5.7 can be adjusted to show the calculated values for this scenario, which has been done in Figure 5.10.

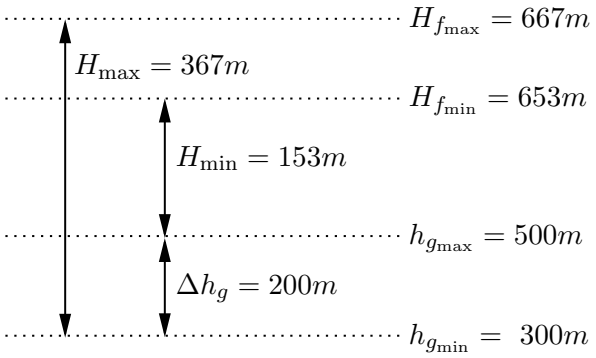


Figure 5.10: Diagram showing the range of flying heights that can be calculated.

Note that all the values along the right-hand side of the figure are absolute heights above sea level. The other three height values are relative to their respective datum lines.

The range of possible flying height (H_f) values is calculated using Equation 5.10. With this range in mind, a height of 660 m above sea level is chosen as the flying height. This is just below the maximum height, so as to favour larger grid cell sizes which correspond to a faster search. The actual maximum GSD and %Overlap for this scenario can now be calculated. The height for both calculations should be taken with respect to the lowest point in the topography ($h_{g_{\min}}$), and is thus 360 m.

The GSD can be calculated using Equation 5.4, and the resulting value is roughly 4.6 cm/px. This is lower than the maximum of 4.7 cm/px and confirms that this is an acceptable flying height.

The %Overlap can be calculated using Equation 5.16. The result is 311%. This is a rather large value, due to the conservative approach to ensure complete coverage. To reduce this value using the proposed discretisation technique, ideally one would want a steeper bank angle or a lower flying speed.

Since this height is above the peak of the mountain, it should be noted that the UAV can safely fly at any point in the map without physical obstructions. However, to ensure complete coverage and target detection (provided the target is present in the environment), certain areas are excluded from the UAV search.

It is assumed that these excluded regions would be better searched manually and that the UAVs would be best utilised outside of these regions. Taking the contour lines from Figure 5.9 and excluding these regions results in Figure 5.11.

Now that the obstacles for this region have been defined, and a flying speed and height have been selected, discretisation can begin. The steps followed

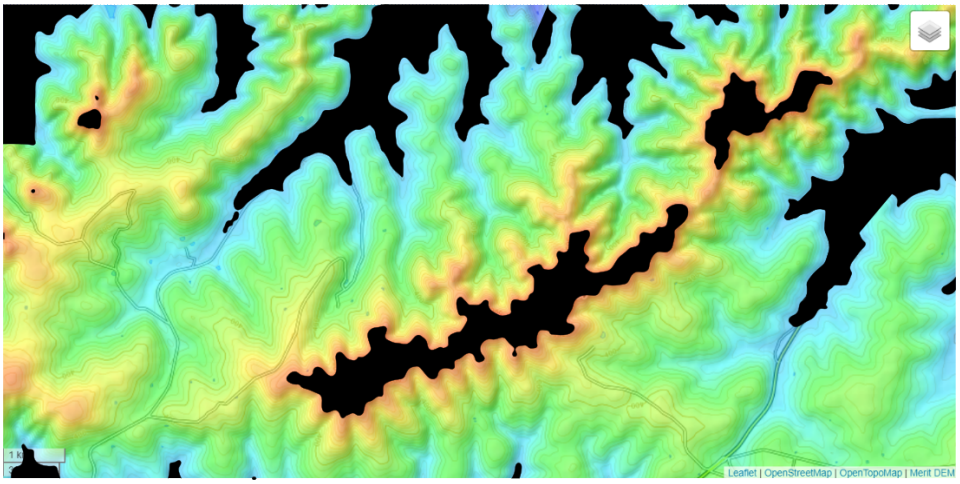


Figure 5.11: Contour map of Spitskop with excluded regions for the search shown in black.

here are listed in Section 5.2.3 and are repeated with the same numbering convention below:

1. The chosen flying height (H_f) is 660 m above sea level. The discretisation is performed relative to the highest point in the topography ($h_{g_{max}}$), which is 500 m, making the relative height from this point 160 m. This height is used to calculate the FOV with Equations 5.2 and 5.3. The resulting FOV is 164 m in the x-dimension (FOV_x) and 110 m in the y-dimension (FOV_y).

2. The chosen flying speed (V_f) is 14 m/s. This can be used to calculate a minimum turning radius using Equation 3.7. The bank angle is assumed to be 25 degrees. This is a generalised assumption that will be used throughout this project unless stipulated otherwise. It is possible that an aircraft may even be able to sustain a 60 degree bank angle, so this is a conservative value. With this in mind the resulting minimum turning radius is roughly 43 m.

3. Now the square discretisation can be calculated according to Equation 5.14, resulting in a side length (l) for the squares of 89.8 m.

4. Lastly it is important to double check that the minimum turning radius fits into the suggested square discretisation. Since half the square discretisation results in a value of about 45 m, this is confirmed.

When adding discrete obstacles to this two-dimensional representation of the environment, it is important to note that the STC algorithm is going to be applied to this discretisation. The STC algorithm will be detailed in Chapter

7, and requires that cells be arranged into groups of four, effectively doubling cell sizes and decreasing overall resolution.

Discrete obstacles are added to the maps at the larger cell size. The larger cell sizes are what is used in the divide-areas algorithm, where cells are assigned to UAVs for searching. Seeing as the UAVs can physically fly over any point in the map, it would be up to the search team to decide whether under- or over-estimating the obstacles would be more appropriate.

Under-estimating the obstacle sizes means the UAV may cover certain cells incompletely and miss a target present in that cell. Over-estimating obstacle sizes means that more areas where the UAVs could potentially detect a target are excluded from the search, but the UAVs can more efficiently cover areas where detection is guaranteed provided the target is present.

In this case, it is assumed that the search team opted for over-estimating obstacles. The resulting map for this is shown in Figure 5.12, which shows the discretisations with the discrete obstacle approximations. Both the smaller cells and larger cells are portrayed in the figure.

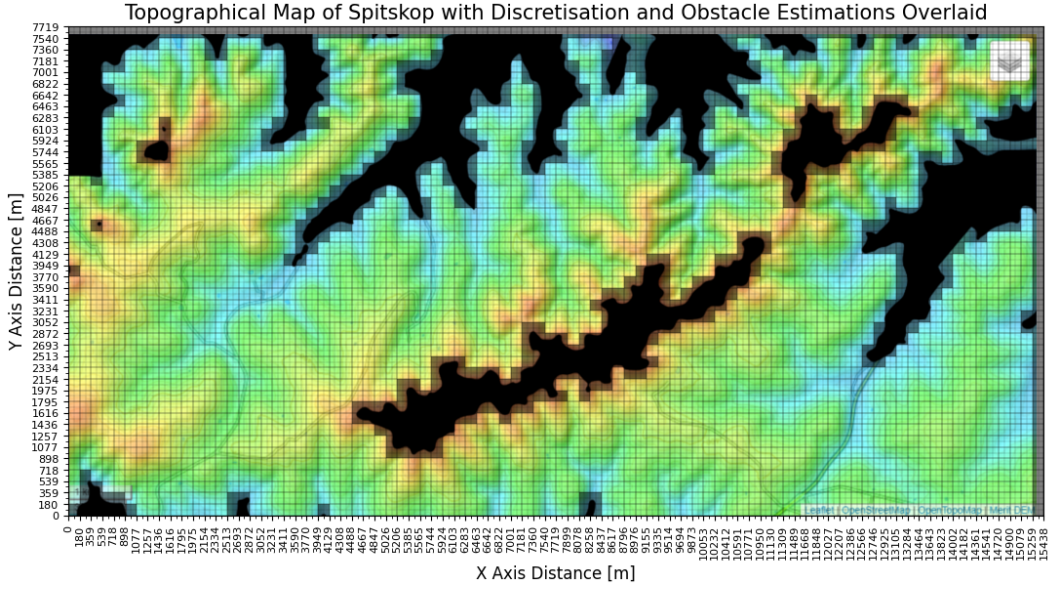


Figure 5.12: Graph showing the map of Spitskop with the discretisations overlaid, including the over-estimated discrete obstacles.

Table 5.3 shows a summary of the parameters used for the discretisation. Note that the flying height (H_f) is a height above sea level. The height used in calculating the discretisation size would be this height relative to the highest point in the topography. In contrast, the height used to calculate the actual GSD and %Overlap achieved would be with respect to the lowest point in the topography.

A more detailed version of this, namely Table A.1, is shown in Appendix A. This version shows a summary of the values and associated equations re-

quired to formulate this discretisation. For each of the examples following this one, a summary table will be given as well as an associated detailed table in the appendix.

Variable Name	Value	Units	Equation
ϕ_{max}	25.0	degrees	
V_f	14.0	m/s	
H_f	660.0	m	5.9
l	89.8	m	5.14
GSD	4.6	cm/px	5.4
%Overlap	311	%	5.16

Table 5.3: Summary of the parameters for the Spitskop environment.

5.4.2 High Altitude Mountainous SAR

The example shown in this section is Champagne Castle, a section of the Drakensberg mountain in KwaZulu-Natal. The highest peak above sea level in this case is 3377 m. Figures 5.13 and 5.14 show a satellite image and topographical map of this area respectively. The actual size of this map is 11.5 km by 5.5 km.

Overall this is a much steeper environment. For this reason a likely scenario for SAR in this case may be hikers that have fallen and are injured near cliffs or other treacherous terrain.

For this example, the Strix 400 UAV will once again be used in combination with the Sony RX1R II camera. The benefits of this UAV for mountainous terrain were discussed in Section 5.4.1. In addition to these benefits, this UAV has a high maximum flying altitude, making it ideal for this high altitude scenario.

Although this environment features a much steeper topography, the maximum allowable variation in topography for this camera and UAV combination is still 214 m. Using the methodology detailed in Section 5.4.1, the region chosen for search is that spanning from an altitude of 3200 m to 3377 m. This represents a topographic variation of 177 m. It is assumed that someone has gone missing near the peak of Champagne Castle, hence the targeted search of this high altitude. Due to the steep environment, this represents a fairly narrow search area. Narrow regions of lower altitude would also be possible. It depends on where a survivor is most likely to be.

This example also features the South Africa to Lesotho border. The assumption here is that the UAVs are not permitted to cross this border. This narrows down the allowable search area even further.

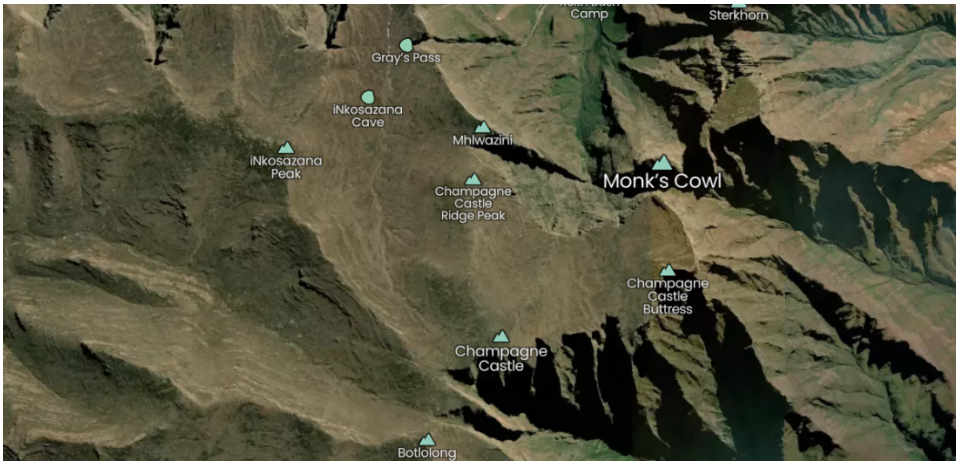


Figure 5.13: Satellite image of Champagne Castle in the Drakensberg. [9]

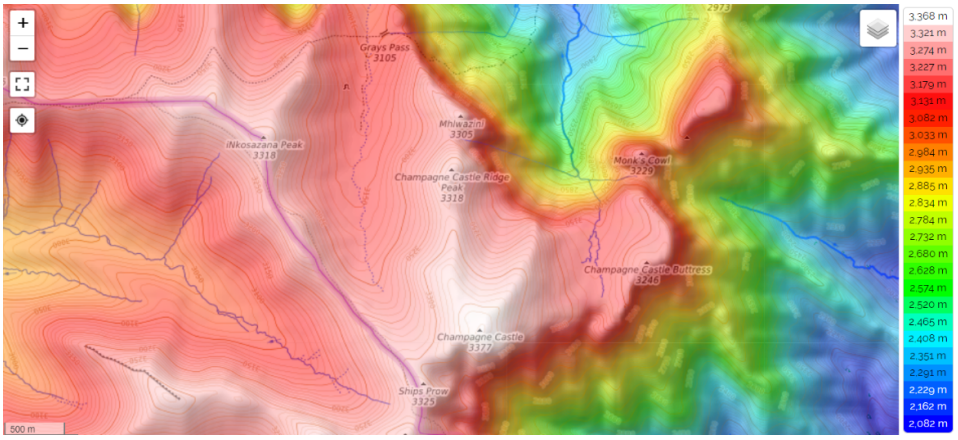


Figure 5.14: Contour map of Champagne Castle in the Drakensberg [8].

A summary of the parameters used for the discretisation are shown in Table 5.4. A more detailed table of the parameters and equations used to perform the discretisation in this example are listed in Table A.2. The range of possible flying heights found can be seen in Table A.2. The flying height is chosen at the lower end of the spectrum to maximise the range of topographic heights wherein a target can be detected.

Note that once again, the UAV can physically fly over any region of the map, since the chosen flying height is 3535 m, as listed in Table 5.4. Crossing the border is however strictly not permitted. The obstacles are therefore set up to over estimate the border obstacle, but under-estimate the rest of the obstacles. This increases the area that the UAVs can search, but still guarantees that the border will not be crossed.

Figure 5.15 shows the continuous obstacles on the topographic map. The Lesotho border obstacle is the obstacle on the bottom left of the diagram. The effective size of the search region for the UAVs becomes much smaller than the

Variable Name	Value	Units	Equation
ϕ_{max}	25.0	degrees	
V_f	14.0	m/s	
H_f	3535.0	m	5.9
l	88.6	m	5.14
GSD	4.3	cm/px	5.4
%Overlap	288	%	5.16

Table 5.4: Summary of the parameters for the Champagne Castle environment.

size of the map for this example. Some regions are also enclosed spaces in this two-dimensional map. How these are dealt with during an actual search will be discussed in Chapter 6.

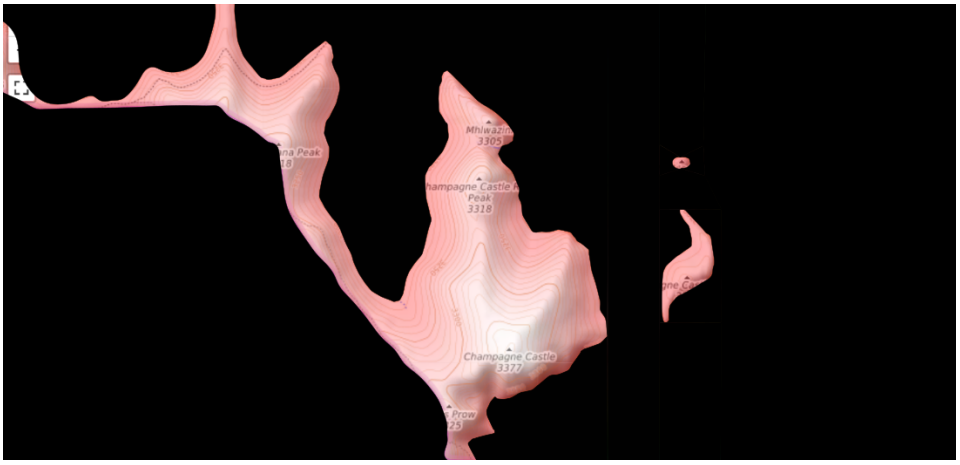


Figure 5.15: Contour map of Champagne Castle with excluded regions for the search shown in black.

Figure 5.16 shows the obstacle approximations in the discrete environment. Note the overestimation of obstacle sizes on the bottom left obstacle, representing the Lesotho border.

5.4.3 Ground SAR

Ground search and rescue often involves people going missing or running away from populated areas. For this reason, the scenario shown in this section is the surrounding area of a town called Aberdeen in the Northern Cape. Figures 5.17 and 5.18 show the satellite and topographic maps of the area respectively. They represent a real world area of roughly 15.8 km by 7.6 km.

For this example, the Wingtra I UAV is used in combination with the Sony RX1R II Camera. This UAV has a slightly higher cruise speed than

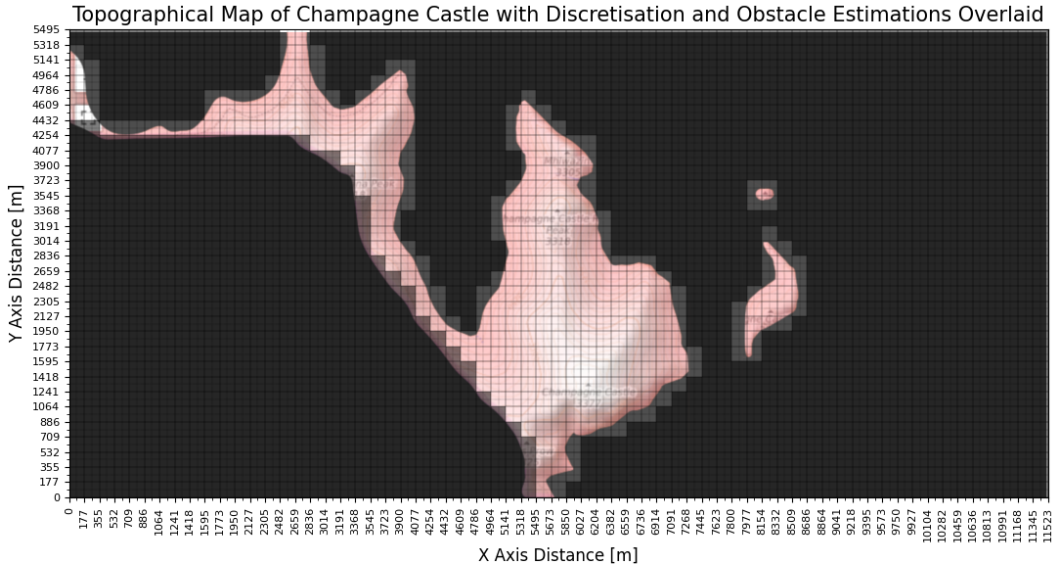


Figure 5.16: Graph showing a contour map of Champagne Castle with the discretisation overlaid, including the discrete obstacle approximations.

Figure 5.17: Satellite map of Aberdeen and the surrounding area. [7]

the Strix 400 UAV used in both mountainous examples. This means the area can be searched faster, while not allowing for as large topographic variations. The topography in this map varies between 706 m and 1711 m. However, it is assumed that the mountain in the top left corner does not need to be searched using the UAVs. The mountain is excluded by excluding any area above 850 m height above sea level from the search. This results in a total environment variation of 144 m, which is within the 167 m limit that results from this UAV and camera combination.

The mountain may be excluded for a number of reasons. One reason could be that a ground team was deployed to search this area instead of using an aerial search with the UAVs. It may also have been determined that this is an

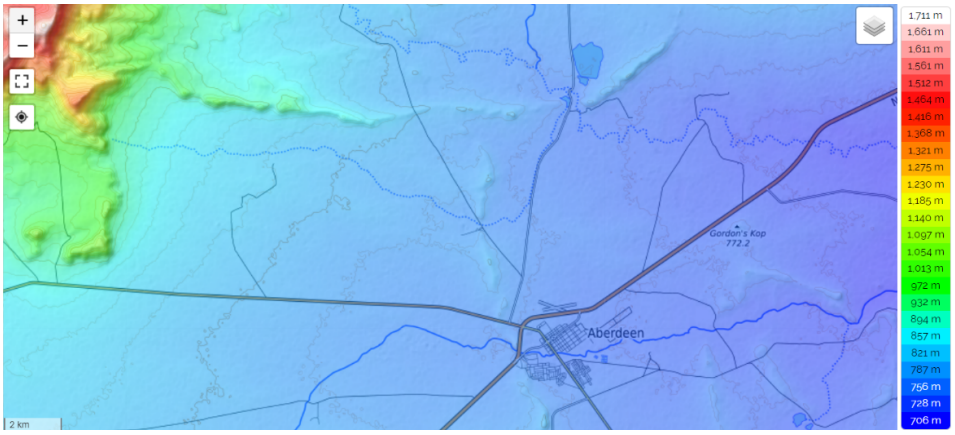


Figure 5.18: Contour map of Aberdeen and the surrounding area. [8]

unlikely location for the survivor to go, therefore the UAV are focused on the larger open area around the town.

The final flying height that is chosen for the craft is 1060 m. This means that the mountain is a physical obstacle at any point where it exceeds this height.

Figure 5.19 shows the excluded region in black and grey. The grey region is the part of the excluded region that is a physical obstruction for the UAVs at their flying altitude. The UAVs cannot traverse this part of the map at the chosen constant flying altitude. The black portion represents a region that can be traversed at the flying altitude, but is simply excluded from the search.

The town of Aberdeen, featured at the lower end of the map will also not be searched. It is assumed that the target will not be present in this area. Figure 5.20 shows the final discretisation overlaid onto the map. In this image, the town of Aberdeen is also excluded using discrete obstacles.

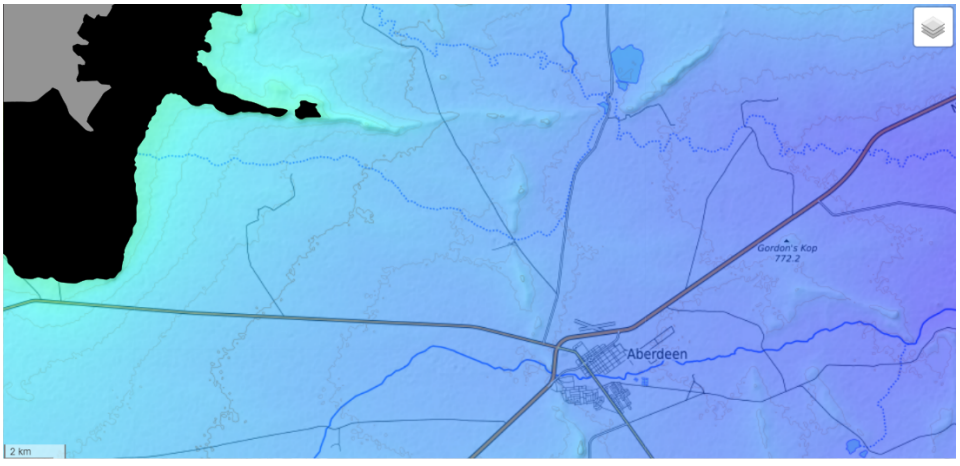


Figure 5.19: Contour map of Aberdeen where excluded regions for the search are shown in black, with the grey portion representing physical obstructions.

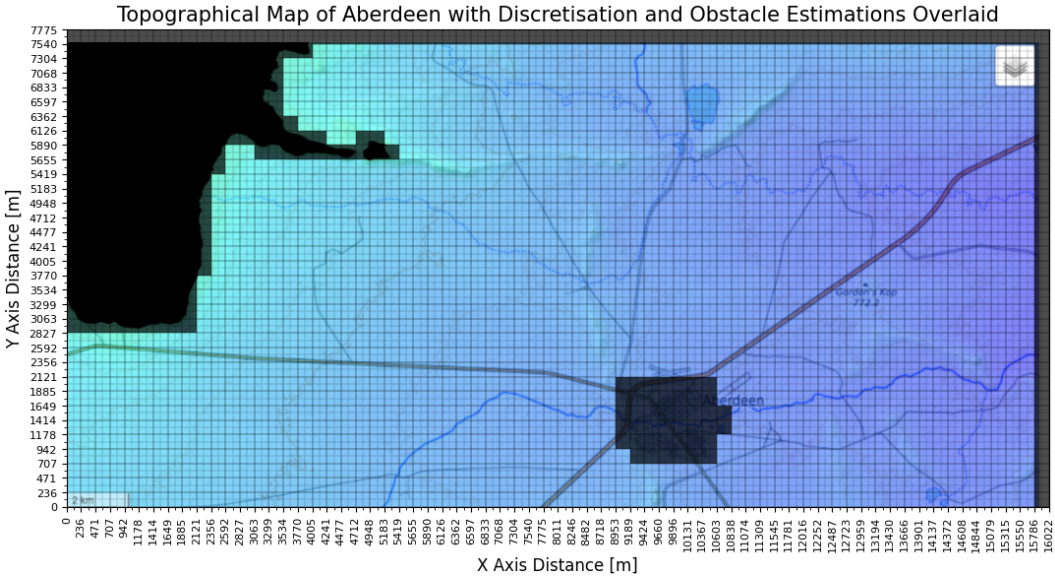


Figure 5.20: Graph showing a contour map of Aberdeen with the discretisation overlaid, including the discrete obstacle approximations.

The excluded region representing the mountain is also converted into discrete obstacles, and it is assumed that the search team has decided to overestimate this region for this scenario.

Table 5.5 summarises the parameters used for the discretisation of this environment. For a more detailed listing of the calculations used to discretise this environment, see Table A.3 in Appendix A.

Variable Name	Value	Units	Equation
ϕ_{max}	25.0	degrees	
V_f	16.0	m/s	
H_f	1060.0	m	5.9
l	117.8	m	5.14
GSD	4.5	cm/px	5.4
%Overlap	208	%	5.16

Table 5.5: Summary of the parameters for the Aberdeen environment.

5.4.4 Marine SAR

Marine SAR may involve vessels in distress or swimmers that have been swept in by rip-currents. The scenario shown in this section is a portion of the Jeffreys Bay coastline in the Eastern Cape. Specifically, the beach shown in a

popular swimming beach called Main Beach. This is a smaller map and the plausible scenario here would be a swimmer swept up by a current, and is now missing.

Figures 5.21 and 5.22 show the respective satellite and topographic maps of the area, representing a real world area of 3.3 km by 1.6 km.

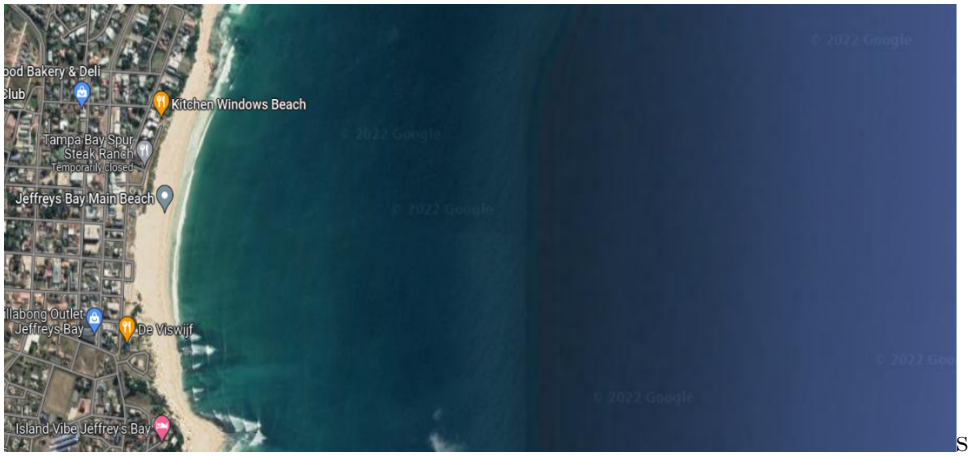


Figure 5.21: Satellite image of Jeffreys Bay Main Beach. [7]

Figure 5.22: Contour map of Jeffreys Bay Main Beach. [8]

The GULL 24 UAV is chosen in combination with the Sony RX1R II camera for this example. This UAV was developed specifically for marine applications and features the ability to take off and land on water. It is therefore viewed as an appropriate UAV for this scenario.

This map features essentially no topographic variation once the coastal land is excluded. It is assumed that an estimated variation of 10 m would suffice. This would account for any slight variations due to rocks or waves.

The data available on the GULL UAV simply states that the cruise speed should be lower than 38 m/s [68]. No data was found regarding a minimum, but it is assumed for this application that 19 m/s would be an acceptable speed. It should also be noted that this craft cannot be on the water when there are waves higher than 0.3 m.

Figure 5.23 shows the map with the coastline excluded as an obstacle, shown in black. With people being swept into the ocean, time is of the essence, hence this application boasts a rather high altitude, high speed implementation. The flying height chosen for this example is 215 m above sea level.

Figure 5.23: Contour map of Jeffreys Bay with excluded coastal region shown in black.

Due to the small environment, a smaller minimum turning radius allows for a more appropriate discretisation. It is assumed that for this smaller environment, the GULL 24 UAV can sustain bank angles of 35 degrees. This may not be the case, and so a different UAV may be more appropriate for this example in a real world scenario

Because the UAV flies above any physical obstructions that may be on the ground, the discrete obstacles in this case are underestimating the excluded coastline region. Figure 5.24 shows the discretisation overlaid on the map, with the few discrete obstacles used to represent the coastline.

Table 5.6 shows a summary of the parameters used for the discretisation of this environment. Table A.4 is a more detailed table, showing the discretisation steps as a set of values and equation references. This can be found in Appendix A.

5.4.5 Discussion

The proposed discretisation technique worked well with the four example environments. The Aberdeen environment is the only one where there is a physical

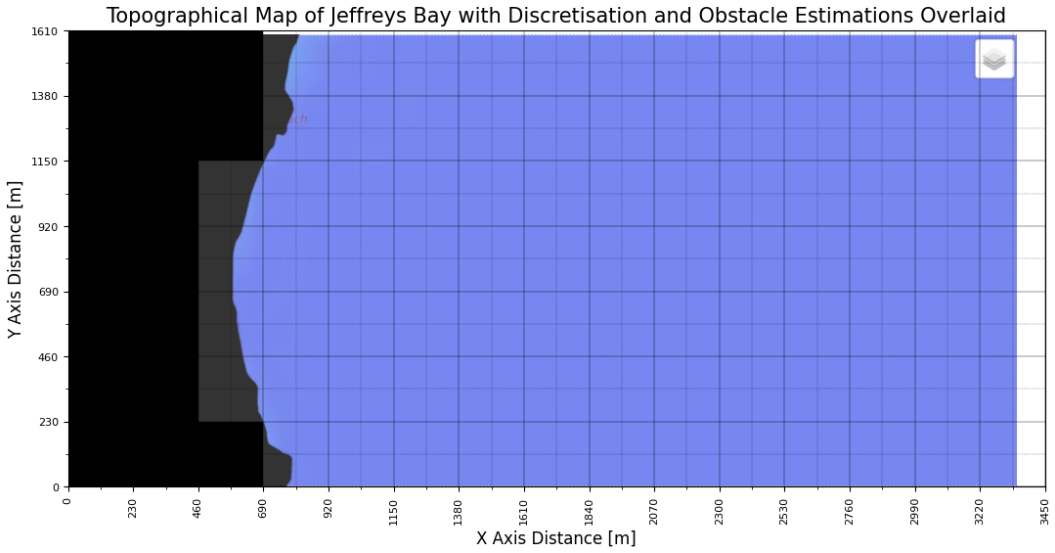


Figure 5.24: Graph showing a contour map of Jeffreys Bay with the discretisation overlaid, including the under-estimated discrete obstacles.

Variable Name	**Value**	**Units**	**Equation**
ϕ_{max}	35.0	degrees	
V_f	19.0	m/s	
H_f	215.0	m	5.9
l	115.0	m	5.14
GSD	2.8	cm/px	5.4
$\%Overlap$	92	%	5.16

Table 5.6: Summary of the parameters for the Jeffreys Bay environment.

obstruction at the flying altitude. In all other examples, the obstacles represent no-fly zones. Regions are also excluded due to the limits of the cameras onboard. The constant flying altitude essentially limits the topographic variation in the search area, so that survivor detection can still be guaranteed. It may be useful to develop a version of this algorithm where the flying altitude of the UAVs can vary, but this is not covered in the scope of this project. In the mountainous examples, the limit in topographic variation that can be accommodated makes for higher obstacle densities. The resulting environment for the Champaigne Castle example also forms enclosed regions.

In all the examples, the flying altitudes are quite high. The resulting dimension of the discrete cells is over 80 m, in each example. This is well above the physical dimensions of UAVs. This means that so long as the UAVs are not flying in the same cells, there will be no collisions between them, according to the UAV collisions model that was given in Section 3.5.

Chapter 6

Divide Areas Algorithm

This chapter describes the approach that was used to divide the search environment into sub-regions based on the individual UAV locations. The approach is based on the divide areas algorithm for optimal multi-robot coverage path planning (DARP). Section 6.1 provides an overview of the DARP algorithm and how it executes. The advantages and disadvantages of DARP are also discussed, as well as the motivation for using the algorithm. Section 6.2 describes how the DARP algorithm was modified for the multi-robot search and rescue problem. Section 6.3 shows some illustrative examples of how DARP is applied to the grid map representations that were fitted to real-world maps in the previous chapter.

6.1 DARP Algorithm

The approach used in this project to achieve coverage is two-fold. First the environment is divided into sub-regions based on the initial positions of the robots. Next, each robot is expected to search a specific sub-region in the environment using a single robot coverage technique. This section discusses the algorithm used to divide the environment into sub-regions.

The algorithm that will be discussed is the divide areas algorithm for optimal multi-robot coverage path planning (DARP). The DARP algorithm was developed by Kapoutsis et al. [6]. The background information presented in this section was sourced from the paper that they wrote discussing their implementation.

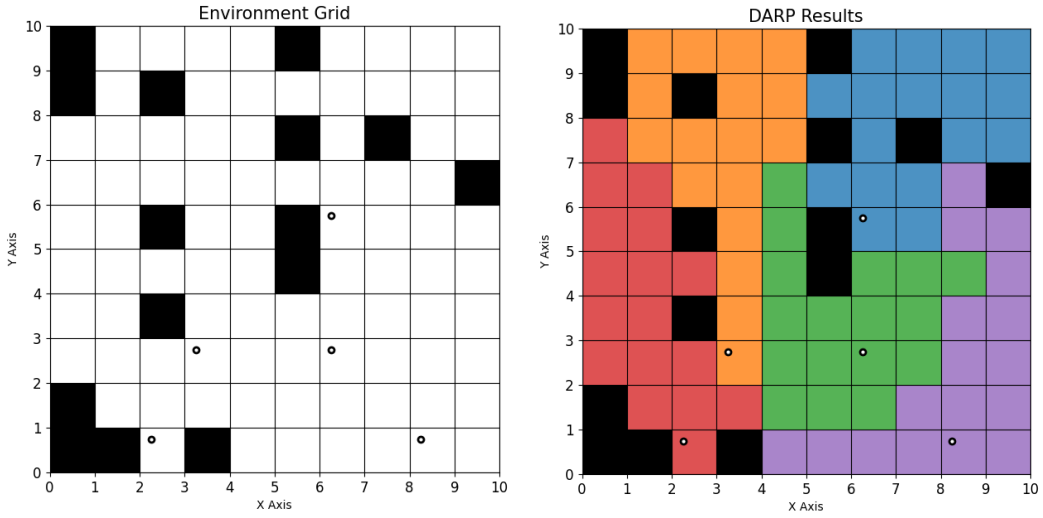
DARP is a grid-based, distributed, and offline algorithm. Since it is offline, the environment is known in full a priori, as well as the robot initial positions. The environment division is then executed to satisfy a number of requirements to optimise the coverage, and is based on the initial position of the robots in the search region.

The distributed nature of this algorithm means that area is divided into sub-regions. However, to achieve optimal coverage, this algorithm proposes

that one robot is assigned to cover each region and that the regions should be contiguous and equal sized.

Because it is a grid-based algorithm, the environment should be discrete and two-dimensional. An example environment is shown in Figure 6.1a. The dimension of this environment is a 10 by 10 grid. This environment also has several discrete obstacles, shown as solid black cells. The rest of the cells can be seen as free cells. The initial positions of the five robots in this environment are shown as black and white dots.

The results of applying DARP to the example environment can be seen in Figure 6.1b. The free cells are grouped to form contiguous sub-regions represented in different colours. Each robot initial position also resides in its own sub-region. An individual area coverage technique can then be applied to cover each sub-region with the robot that starts within that region.



(a) Environment Grid (b) DARP Application

Figure 6.1: Example environment grid showing the resulting area division after applying the DARP algorithm to it.

The objectives of the DARP algorithm are listed below:

1. Every cell in the environment, that is not classified as an obstacle, must be covered. This is known as complete coverage.

2. Each cell in the environment must only be visited once, and only by one of the robots. This is known as the non-backtracking requirement.

3. Each robot should have as close to an equal amount of cells as possible assigned to it for covering. Their sets of cells should be of roughly the same size.

4. The sets of cells assigned to each robot should be a connected sub-region. This means that when generating a path to cover the cells within its set, a robot would not need to traverse the sub-region of another robot to reach its own.

5. The initial position of each robot should be contained within the set of cells assigned to it. This means that a robot would not need to traverse another robot's sub-region to reach its sub-region for coverage.

The objectives are met using an iterative approach. The algorithm slowly changes the sub-regions to have spatial connectivity and be of equal size. The location of these regions are based on the robot initial positions, to ensure one is present to search each region. There are therefore an equal number of robots to sub-regions.

Provided each cell is assigned to a robot, each cell will be covered with the single robot coverage algorithm. Note that the no backtracking requirement needs to be met by the single robot coverage algorithm as well.

In each iteration of the algorithm, three multipliers are calculated to update a distance matrix. Eventually it converges until the conditions for optimal coverage are sufficiently met. What the algorithm does in each iteration is summarised below:

1. Assign each cell with coordinate (x, y) to a robot using

$$A(x, y) = \underset{i \in 1,...,n_r}{\operatorname{argmin}} E_i(x, y) \tag{6.1}$$

where A is the assignment matrix, i is the robot index, and E_i is the evaluation matrix for the i^{th} robot.

2. For each i^{th} robot where $i \in 1, ..., n_r$:

   a) Determine the amount of cells assigned to each robot (k_i).

   b) Update correction multiplier (m_i) using

$$m_i = m_i + c(k_i - f) \quad \forall i \in 1, ..., n_r \tag{6.2}$$

   where c is a positive constant, and f is the desired number of cells to be assigned to each robot.

   c) Establish a matrix representing all cells that are spatially connected to the robot (R_i) and one representing those cells that are not connected (Q_i).

   d) Calculate connectivity matrix (C_i) using

$$C_i(x, y) = \min(||[x, y] - r||) - \min(||[x, y] - q||) \quad \forall r \in R_i,\ q \in Q_i,\ i \in 1, ..., n_r \tag{6.3}$$

where r is an index for spatially connected cells in R_i, and q is an index for unconnected cells in Q_i.

e) Generate the randomized matrix (Z_i).

f) Update evaluation matrices using

$$E_i = m_i(E_i \circ C_i \circ Z_i) \quad \forall i \in 1, ..., n_r \tag{6.4}$$

where m_i is the correction multiplier, C_i is the connectivity matrix multiplier and Z_i is the randomized matrix multiplier for each i^{th} robot.

The algorithm starts by operating similar to a Voronoi partition [3]. A matrix is made for each robot with the same dimension as the environment. These are referred to as evaluation matrices (E_i). They contain the distances from each cell in the environment to the respective robots. The original algorithm makes use of the euclidean distance measure for constructing these matrices.

The evaluation matrices are then compared to one another to generate an assignment matrix (A). Each free cell in the environment is assigned the index of a robot. This is done according to Equation 6.1, which initially means that a cell is assigned to the robot closest to it.

The robots are expected to cover the cells that are assigned to them. Because each cell is assigned one robot, the first objective for optimal coverage is met. However, the regions formed by these assigned cells are not necessarily contiguous, nor are they of equal size.

The evaluation matrices are therefore iteratively updated to meet the third and fourth objectives for optimal coverage. This is done using three different multipliers, and the process is illustrated in detail in Figure 6.2.

The first multiplier is the correction multiplier (m_i), which is associated with the third requirement of equal sized regions. A scalar multiplier is calculated for each robot, to be multiplied with its associated evaluation matrix. The calculation of this multiplier is highlighted in green in Figure 6.2.

For each robot, the desired number of cells to be assigned, denoted by f, is the total cells in the region divided by the number of robots. The actual number of cells assigned to the i_{th} robot is denoted by k_i. The difference between these, multiplied by a positive constant (c), is used to to update the correction multiplier according to Equation 6.2.

Overall, the multiplier's value decreases if too many cells are assigned to this robot and increases if too few are assigned. This update occurs in each iteration.

The next multiplier is the connectivity matrix (C_i), which is associated with the fourth objective of contiguous sub-regions. The algorithm detects any regions that are not spatially connected to the robot assigned to it. For each robot there is a matrix representing all the cells that are spatially connected

to the robot they are assigned to (R_i). Similarly there is matrix representing all the cells that are disconnected from the robot they are assigned to (Q_i).

Equation 6.3 shows the equation used to calculate the multiplier. This formula is applied to every cell in the environment. The distance of each cell to the nearest cell within the spatially connected region is calculated and the distance to the nearest disconnected cell is then subtracted from this.

The connectivity matrix increases the likelihood of cells closer to the connected region being assigned to that robot. Those further away in turn become less likely to be assigned to this robot in the next iteration. A weight is associated with this matrix which increases or decreases its effect per iteration. The calculation of the connectivity multiplier is highlighted blue in Figure 6.2.

The algorithm terminates when two conditions are met. The first is that all regions are connected, and so all the connectivity matrices are a matrix of ones. The second is that all robots have been assigned a similar number of cells.

The maximum allowable discrepancy in assignments (Δ_{max}), as shown in Figure 6.2, can be set by the user. This is defined as the maximum allowable difference between the most number of cells assigned to a single robot and the least assigned to a single robot.

The last multiplier is a random matrix multiplier (Z_i), which is highlighted yellow in Figure 6.2. This multiplier is a matrix of small random values that account for any edge cases. Generally, the effect it has is that cells which are equidistant from two or more robots are randomly assigned to different robots until the requirements are met. Distance in this case refers to the adjusted distances of the evaluation matrices.

This multiplier helps resolve edge cases but causes a random element to the solution, meaning the same environment may have multiple different solutions. How much they differ would depend on the weight of the random element, which can be adjusted.

Equation 6.4 gives the equation to update the evaluation matrices in each iteration, with i representing the robot index and n_r representing the total number of robots. This equation is reiterated in Figure 6.2, which portrays the logical flow of the DARP algorithm overall.

The final requirement of no backtracking was met by dividing the environment into smaller cells. Each discrete cell in the environment is divided into four equal sized, smaller cells. These are the cells that the UAVs would traverse. The size of the smaller cells would be determined by the footprint of the UAV's camera to ensure complete coverage. This only works for certain individual area coverage algorithms. The sweep-like search of the spanning tree algorithm which is used in this project means that the no backtracking requirement is indeed met using this technique.

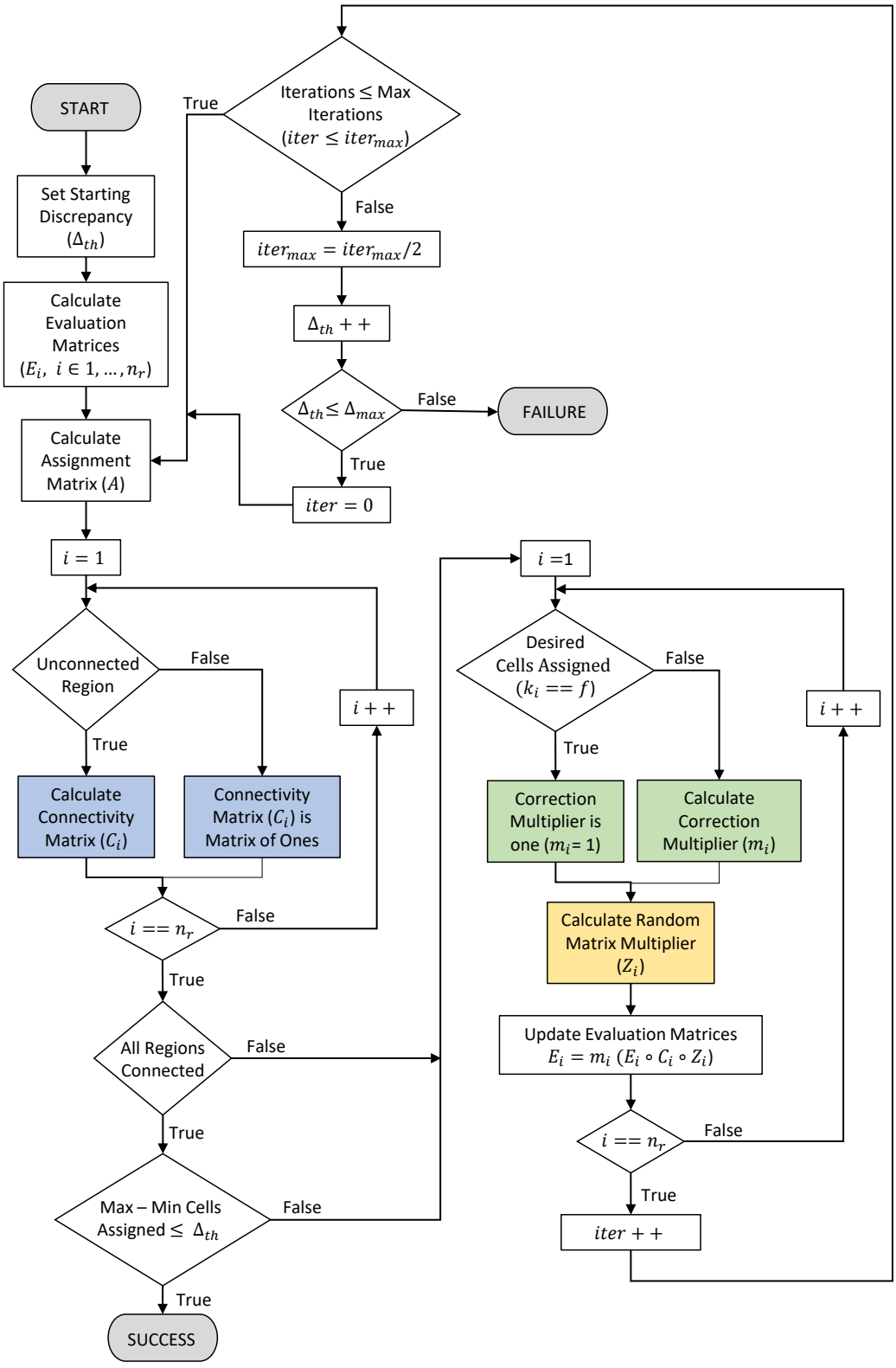


Figure 6.2: Flow diagram representing the logic for the DARP algorithm.

6.2 DARP Advantages and Disadvantages

There are some clear advantages and disadvantages to this algorithm. Firstly, when looking at the fact that DARP is a grid-based algorithm, there are resolution limitations. Many existing algorithms make use of a grid-based representation of the environment. In offline approaches, this is convenient to represent the environment. One disadvantage is that if the grid resolution limits how accurately the environment is portrayed. This is why it is called an approximate approach [17].

This inaccuracy can be combated by introducing some sensor overlap, or by simply ensuring the resolution is very high. However, higher resolution means more computational power is required, so this should not be done irresponsibly.

A cell in DARP also represents four smaller cells, which are representative of the actual discretisation of the environment for coverage by a UAV. This technique to avoid backtracking effectively reduces the resolution, causing the environment representation to be less accurate. The advantage of no backtracking is favoured over the decrease in resolution for the purpose of this project, though.

A search team would need to decide for any given obstacle how best to represent it in a discrete environment. The examples of Section 5.4 showed how a discretisation can be executed and how obstacles can reasonably be approximated, already using the larger cells. In these scenarios there is significant overlap, which would most likely offset any inaccuracies in environment representation.

One drawback of DARP is that it cannot account for regions enclosed by obstacles. Because it is a two-dimensional implementation, it does not allow for the traversing of obstacles by the UAVs. The search area must be a contiguous region of cells and cannot have isolated sub-regions. To address this, Baras et al. [56] proposed a method to extend the algorithm to the third dimension, but this will not be explored in this project .

Since the UAVs fly at a constant altitude, a two-dimensional representation is very intuitive. A higher altitude flight also means that the UAVs are unlikely to encounter any physical obstacles, making it easier to represent the environment in this way.

Because DARP is a distributed algorithm, each UAV travels only within its allocated sub-region. These regions do not overlap, meaning that the UAVs will never collide, provided they follow their planned paths. Therefore, it removes a layer of complexity that often gets added with multi-robot approaches. So long as the UAVs follow their paths within a reasonable margin of error, they will never collide with one-another.

The individual sub-regions also mean that existing single-robot coverage techniques can be used to achieve coverage of the entire environment. The problem of optimising multi-robot coverage is greatly simplified in this manner.

DARP is also an offline approach to coverage path planning (CPP), so the environment is known prior to the planning phase. Online approaches are generally considered appropriate for dynamic environments [43]. This could refer to any amount of objects in motion within the environment that need to be avoided. Often times, the other UAVs in the environment are modeled as dynamic obstacles from the perspective of a single UAV. With this distributed approach, this is unnecessary, because the UAV paths will never cross one-anothers' paths.

Sometimes, online approaches are deemed appropriate because inputting the environment into the planner a priori is considered too costly [43]. In the case of SAR, it is reasonable to assume that plotting the environment from a bird's eye view is possible with available topographic maps and satellite imagery, particularly when flying at higher altitudes. The advantage of having this information available a priori is that the search of the area can be optimised beforehand and that no decision-making is required in real-time during flight [43].

There is generally enough information to map out the aerial environment beforehand, seeing as UAVs fly high enough to avoid most obstructions in an environment. The examples of Section 5.4 illustrated this phenomenon.

It should be noted that in online approaches, information sharing becomes quite critical. If the UAVs have a large enough communication range they can follow a centralized approach, where all data is sent to a central command station or UAV, that plans all their paths simultaneously. They could also all operate independently and simply share information when in range, which is a decentralized approach. [36]

For the offline case, approaches can generally be viewed as centralized. Their paths are planned in full beforehand, and the only thing that may require monitoring during flight, is whether they follow this trajectory within an acceptable margin of error to avoid collisions, and achieve good coverage. For SAR it would also be beneficial to receive real-time data that can be used to locate a target.

A benefit of the offline approach is that the UAVs require less sensory equipment on board. If a UAV is lost during a search, which is not unlikely in treacherous environments, it is best to reduce the loss as much as possible. It is also possible to continue without connectivity to a central command station, provided the UAV has a navigation controller on board and can follow its planned trajectory. This may be useful in harsher weather conditions.

The methods described in this thesis work on the assumption that the UAVs have a guidance controller on board and that the UAVs can follow their paths within a reasonable margin of error.

6.3 Algorithm Modifications

A minor modification was made to the DARP algorithm to make random environment generation easier. An enclosed space checker was introduced to remove any regions enclosed by obstacles that cannot be reached by the UAVs. This ensures that the algorithm will not fail when a randomly generated environment contains enclosed regions. The obstacle density would just change, since the enclosed regions are set as obstacles themselves.

Another change was introduced regarding the distance measures used. The DARP algorithm originally only has the option of using euclidean distances to set up the evaluation matrices. The option to use manhattan or geodesic-manhattan distance measures was introduced into the algorithm. Depending on the weight of the random matrix multiplier, these distance measures tend to change the general shape of the sub-regions generated by DARP. These distance measures in particular were also used in a paper by Nair and Guruprassad utilizing the Voronoi partition to divide an environment [3]. In this, they compared the various methods and chose to use the geodesic manhattan method for their final implementation. They found that manhattan distance measures worked better for a grid-based environment and that geodesic distances made it easier to divide the environment around obstacles in a logical manner. Because of this it was deemed appropriate to include the manhattan and geodesic manhattan options for further investigation in this context as well.

Another modification that was made to the DARP algorithm was that initial positions of the UAVs were redefined for use with multiple UAVs in a search and rescue context. An initial position was reinterpreted as a point from where the UAVs start their search, already at the correct altitude. This implies that there needs to be a take-off and landing region for the UAVs. For this project, it is assumed that the UAVs take off from a central ground station and fly to their initial positions to execute their search. These initial positions may be spread out over the map, or clustered closely together. The search team can choose a configuration that applies well to the environment at hand. A series of configurations were proposed for different numbers of UAVs.

The DARP algorithm was also modified to take into account the endurance of the UAVs. The path lengths of the UAVs should not exceed their flight capabilities. A more detailed description of how the DARP algorithm was modified for refuelling will be presented in Chapter 8. How this is done using DARP is by assigning multiple sub-regions to a single UAV. If refuelling is required, a UAV would finish a particular sub-region, land, and then take off again to a new initial position for searching the next sub-region that is assigned to it.

It should be noted that the ground station, where take-off and landing occur, is seen as an obstacle. It would be unwise to allow UAVs to fly over this area seeing as it may induce collisions. It is also unlikely a survivor would

be present in such an area without being found.

The original algorithm assumes an orthogonal vehicle when implementing their sub-region coverage technique. This means that the paths for cell coverage are always straight lines of the same length, since the cells are square and the use Manhattan motions. With this in place, if the algorithm can generate sub-regions of exactly the same size, the path lengths of the individual UAVs would be the same length. This is assuming these single robot coverage algorithms cover each cell only once.

For this project, the dynamics constraints of the UAVs were introduced in the sub-region coverage technique, and the semi-circular path segments are actually slightly shorter than the straight-line segments. Strictly speaking, the number of rotations in a path would influence the total path length. However, this slight discrepancy in the total path lengths was considered to be negligible, and was not taken into account in the sub-region division.

6.4 Illustrative Examples with Different Environments

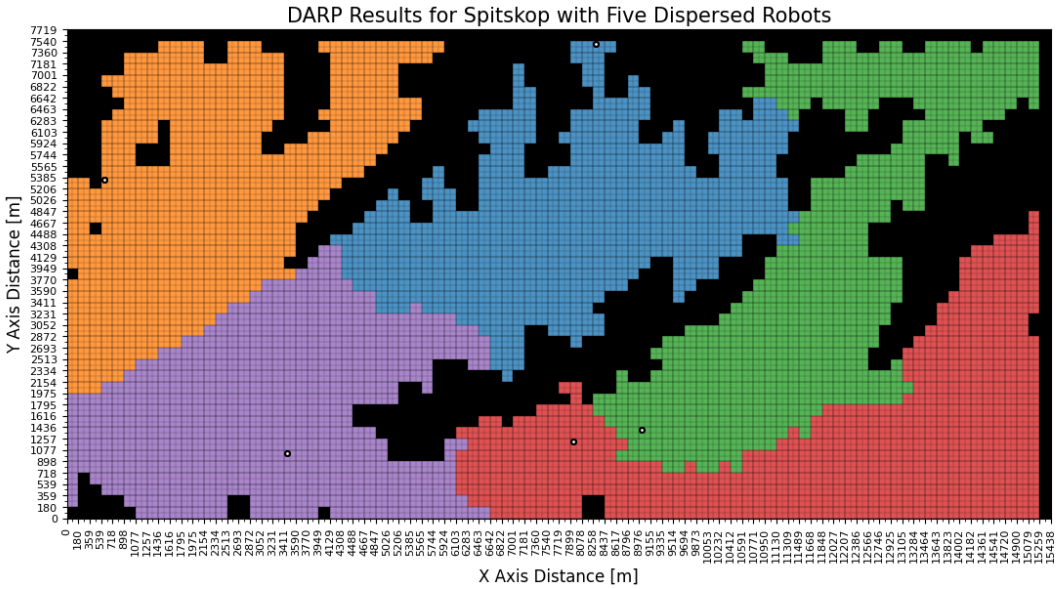
This section shows examples of how the divide areas algorithm divides the demarcated search area into sub-regions, using the same four environments that were discretised in the previous chapter. For clarity, the sub-region divisions are shown on the discretised environments and they are not overlaid on the original satellite images or topographical maps. For each example, a different number of UAVs are used to search the area, to illustrate the effect of the number of UAVs.

Two separate UAV configurations are also explored for each scenario. A dispersed configuration means that the UAVs are some distance apart and are fairly spread out through the environment initially. A clustered configuration implies that the UAV initial positions are very close together, which tends to pose more of a challenge when attempting to divide areas fairly. The clustered scenario is a precursor to the configurations that will be used when a central take-off and landing zone is explored in more detail. Due to the central take-off, the UAVs are likely to be closely spaced once they reach their initial positions at the search altitude.

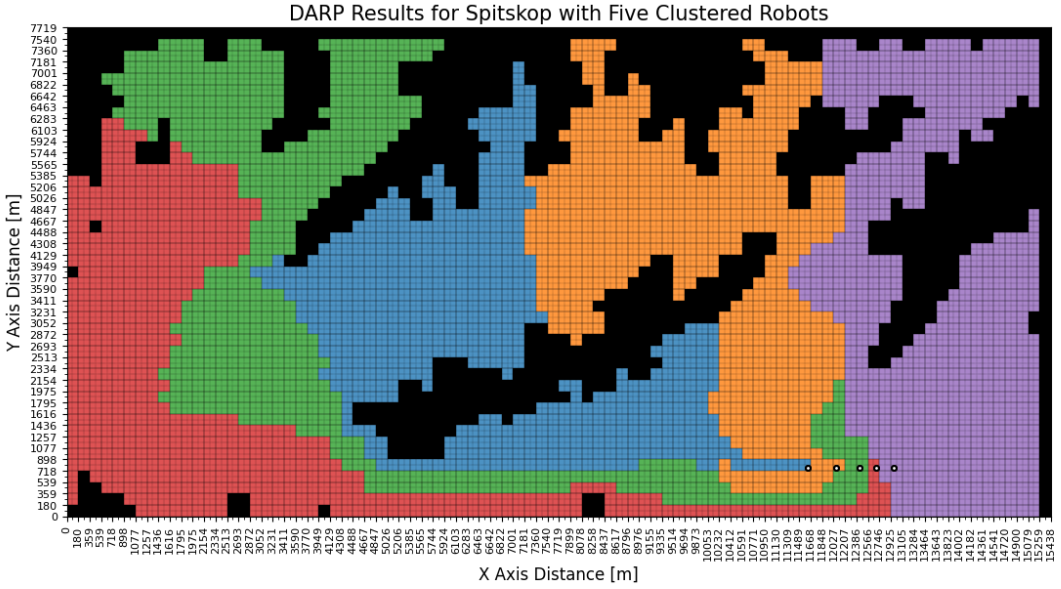
Spitskop

The Spitskop environment is shown in Figures 6.3b and 6.3a. Due to the large area to be covered, five UAVs are used. The UAVs are assumed to start within the centre of one of the small discrete cells and are shown on the figures as black dots with a small white dots at their centres.

In the clustered example, the UAVs are spaced at most five cells apart. This showcases the trade-offs that DARP can make, even when the UAV ini-



(a) Dispersed UAVs



(b) Clustered UAVs

Figure 6.3: Results of applying the DARP algorithm to the Spitskop example environment with five UAVs

tial positions are close together, to form contiguous regions for searching. The black regions in the figures are naturally the discrete obstacles in the environment. The other coloured regions all respectively represent a sub-region to be searched by a particular UAV, which is also present within that sub-region.

The shapes of the sub-regions are clearly very dependent on the UAV initial positions. The regions in the clustered example form very narrow sec-

tions around the UAV initial positions. This is necessary to keep the UAVs in their assigned sub-regions, while also keeping the regions contiguous and equal-sized. In the dispersed example, the sub-regions have shapes similar to what a Voronoi partition would have resulted in.

The size of the Spitskop map is 3698 cells in total, with 967 of them being obstacles. The environment therefore has an obstacle density of about 26%, and a resulting 2731 free cells. With five UAV initial positions, a perfect division of this environment would result in a discrepancy of one. The discrepancy is the difference between the maximum number of cells assigned to one UAV and the minimum number of cells assigned to one UAV. It is a measure of how equally the area is divided. A perfect division will always have a value of one or zero.

The clustered scenario achieved a perfect discrepancy, with 546 being assigned to each robot except one, which has 547 cells assigned to it. The dispersed scenario scenario came close to a perfect division, with a final discrepancy of four cells. The cell assignments for the five UAVs were 546, 544, 547, 548 and 546 cells respectively. In both cases, the resulting sub-regions were very close in size and were contiguous. The objectives for optimal area division were practically achieved.

Champaigne Castle

The next environment is the Champagne Castle scenario. This environment has a high obstacle density, in part to show how the algorithm would deal with a narrow search region. The original discretisation of this environment has enclosed spaces. These enclosed regions can be seen in Figure 6.4.

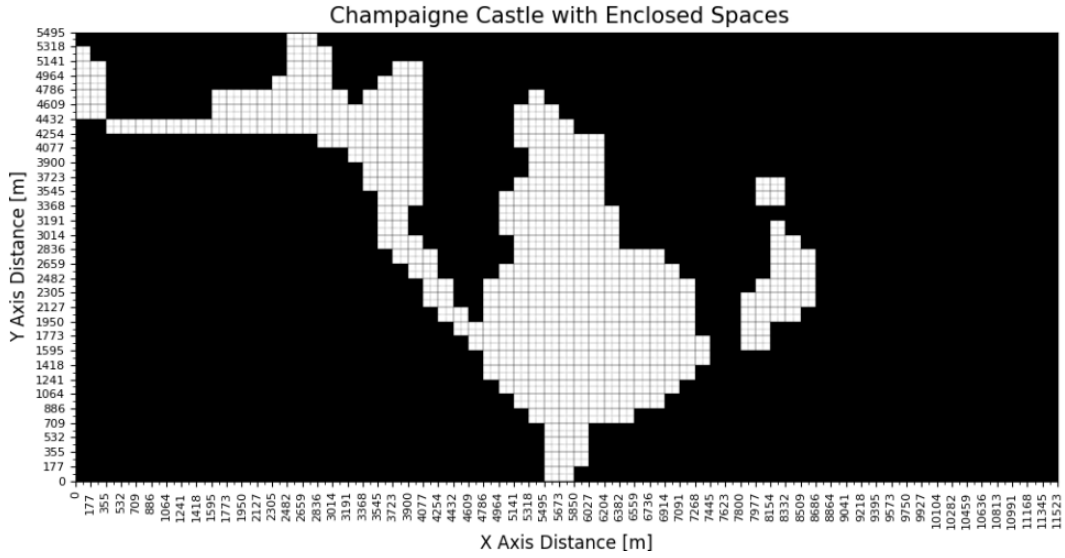
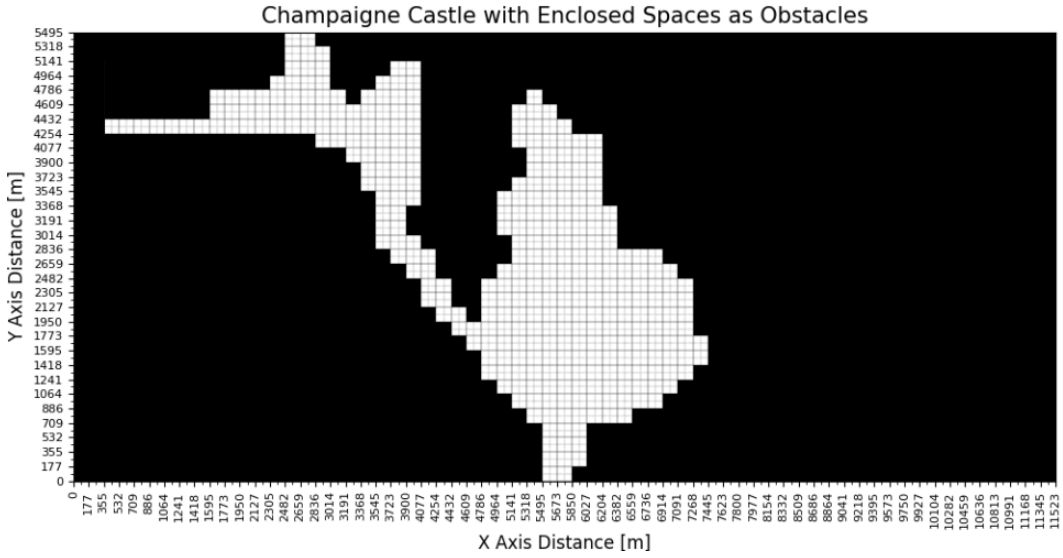
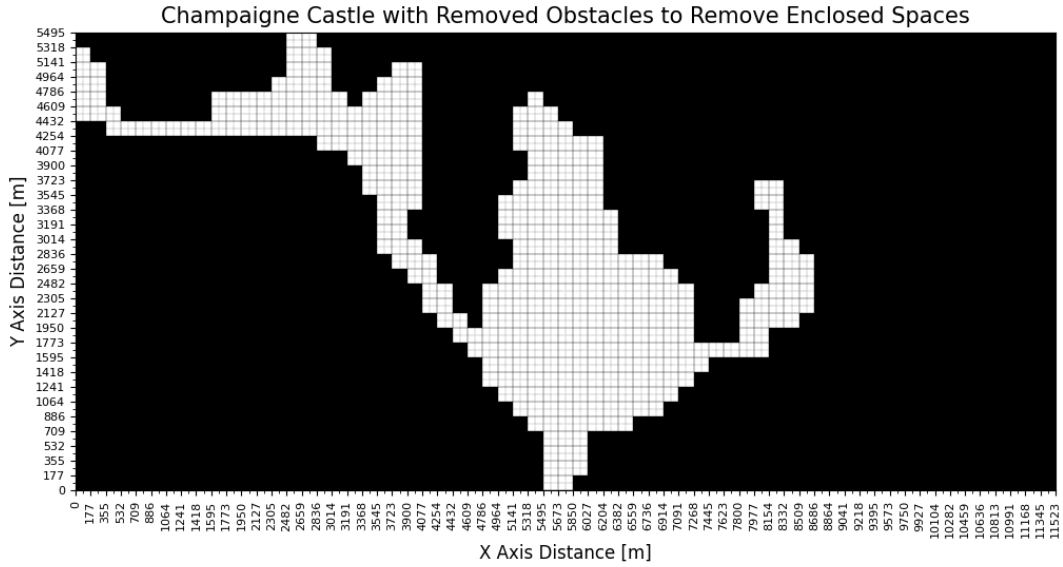


Figure 6.4: Champagne Castle example environment with its enclosed spaces.

There are two options for dealing with these enclosed regions, both of which are shown in Figure 6.5.



(a) Champagne Castle example environment with its enclosed spaces removed as obstacle.



(b) Champagne Castle example environment with its enclosed spaces joined to the main region.

Figure 6.5: Representation of the two ways to handle enclosed spaces on the Champagne Castle example.

The first option is to remove the enclosed regions, that are not spatially connected to the main region, by making them obstacles. Figure 6.5a shows how these regions are ignored by viewing them as obstacles. In this example,

three regions are automatically removed by the algorithm. This is shown by adding black squares in these regions, which represent obstacles. What is left is a spatially connected, but smaller search region. Some regions are effectively not covered.

The second option for dealing with the enclosed regions is to connect them to the main region. This is only feasible for scenarios where the obstacles that get removed are not physical obstructions. For the Champagne Castle example, the obstacles in the environment are not physical obstructions. Some of the obstacles represent the South African and Lesotho border. These obstacles cannot be removed.

The rest of the obstacles in this example can be removed though, since they represent regions that are excluded simply because target detection is not guaranteed in them. The UAVs can technically fly over these regions, therefore a minimal amount of obstacles are removed to allow the UAVs passage. The resulting environment map can be seen in Figure 6.5b. Some of the black squares, representing obstacles, have been removed to make the search zone a spatially connected area.

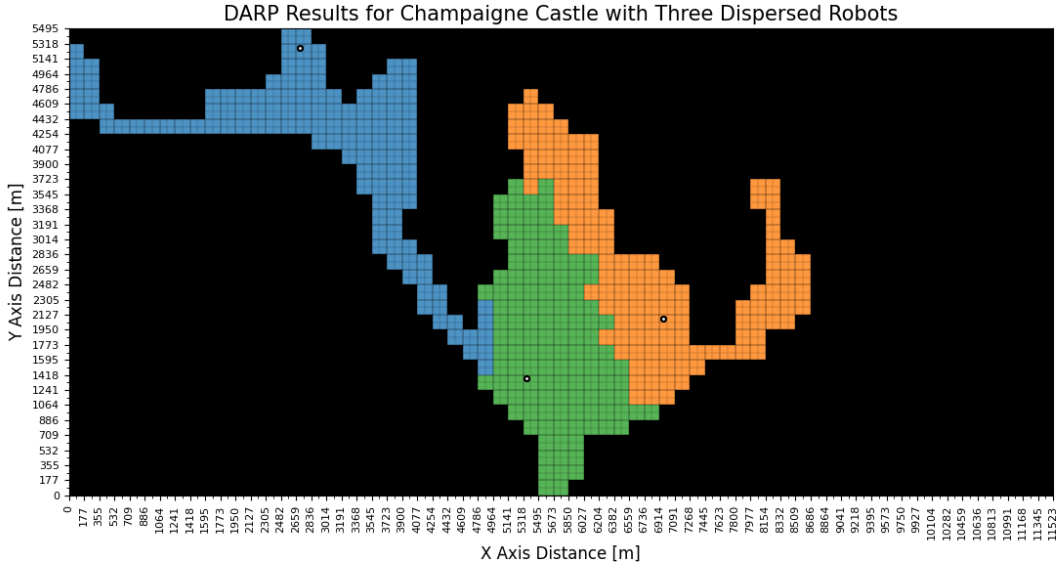
The UAVs can traverse these regions in this case, but are not guaranteed to detect a survivor when flying over them. These areas should therefore not be considered covered in the SAR operation. Some redundant coverage is favoured over missing certain areas for searching in this case. There may be scenarios wherein a search team decides it is more appropriate to exclude these regions than incur redundant coverage by including them. It would depend on the scenario.

The environment representation with the removed obstacles is used in the DARP implementation for this example. The dispersed and clustered versions of this implementation can be seen in Figures 6.6a and 6.6b respectively. For this environment, the sub-region shapes do not differ as greatly, due to the limited space to begin with.

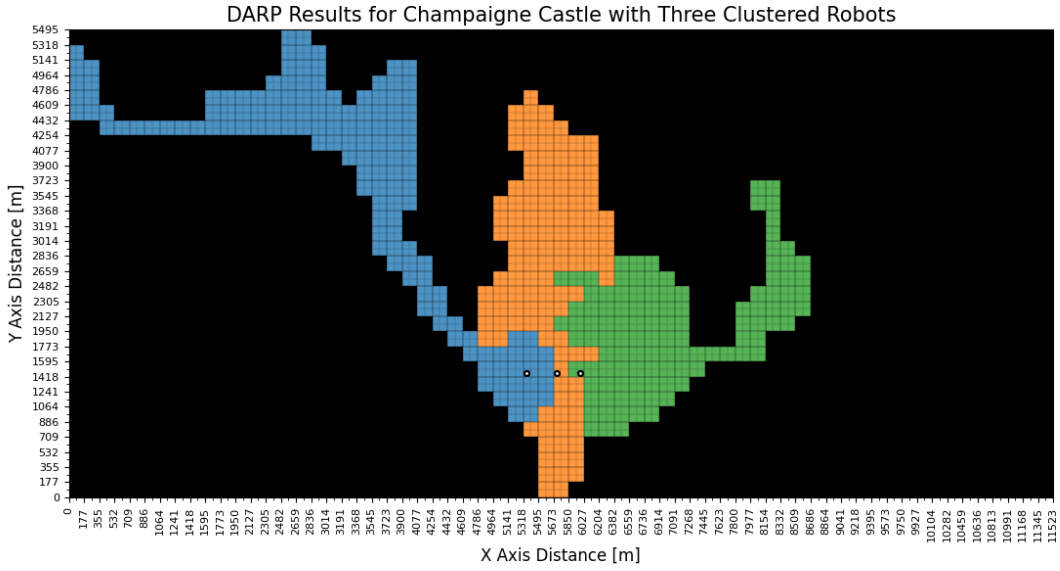
Three UAVs were placed in the environment since it is a smaller search region. The narrow search region means that there are some spaces that are only one large cell wide. These regions make it possible for UAVs to effectively block one another during the division of areas. A UAV initial position should ideally not be in or near such a narrow region.

The overall size of this environment is 2015 cells. Of these, 1641 cells are classified as obstacles, making the resulting obstacle density about 81%. The resulting 374 cells need to be searched by three UAVs, therefore a discrepancy of one would indicate a perfectly equal area division.

Due to the high obstacle density and the challenging environment shape, the area division in this environment is less than perfect for both cases. The dispersed scenario had a resulting discrepancy of 41 cells. The actual assignments for the three UAVs were 123, 146 and 105 cells. For the clustered scenario, there was a discrepancy of 30 cells. The final assignments were 142, 120 and 112 cells respectively.



(a) Dispersed UAVs



(b) Clustered UAVs

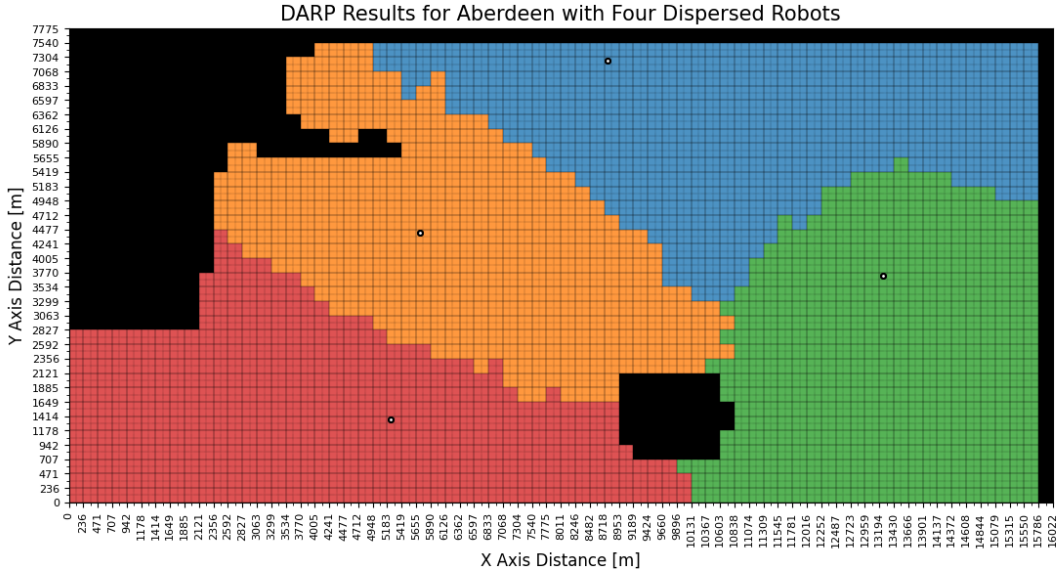
Figure 6.6: Results of applying the DARP algorithm to the Chapaigne Castle example environment with three UAVs

This is not an ideal scenario for application of the DARP algorithm. For the algorithm to have more suitable cell assignments, it may be useful in future to allow some overlap between the individual sub-regions, but this would require collision avoidance to be considered.

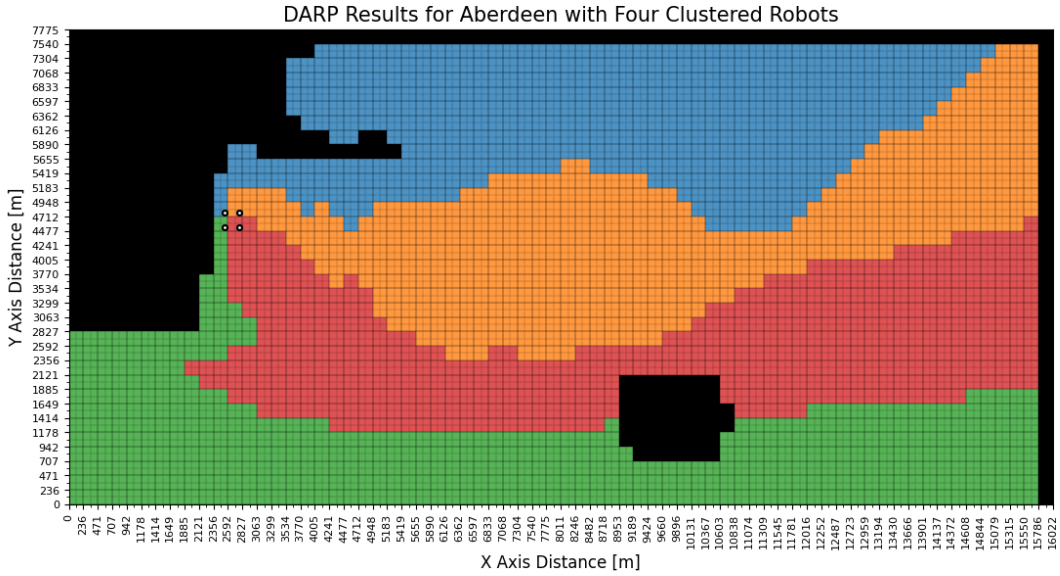
Aberdeen

The third example for a DARP implementation is that of Aberdeen, and in this case four UAVs are used. This is a wider open region and the shapes produced by the DARP algorithm are very intuitive in Figures 6.7a and 6.7b.

The clustered example features a configuration where the UAVs are in a square formation close to an obstacle in the environment. This also results in



(a) Dispersed UAVs



(b) Clustered UAVs

Figure 6.7: Results of applying the DARP algorithm to the Aberdeen example environment with four UAVs

the sub-regions forming very narrow portions near the UAV initial positions. The overall shapes are not quite as elongated as with the Spitskop example. This is because there is more open space for the search, with less obstacles in the environment.

The dispersed example very closely represents a Voronoi partition due to the spread of the UAVs. They are spaced fairly equidistant from one-another, but still far apart.

This Aberdeen map consists of 2244 cells in total, with 394 of them being classified as obstacles. The obstacle density is about 18%. As expected, this is a considerably lower obstacle density than the mountainous environments of Spitskop and Champaigne Castle. With four UAVs, and 1850 free cells, a perfect division would once again result in a discrepancy of one.

For both the dispersed and clustered scenario, a perfect division was achieved for this environment. In both cases, two UAVs were allocated 463 cells and the other two were allocated 462 cells. This environment is notably less complex than the mountainous examples, which is likely why these perfect divisions were achievable. This is an example of an environment to which the DARP algorithm is well suited.

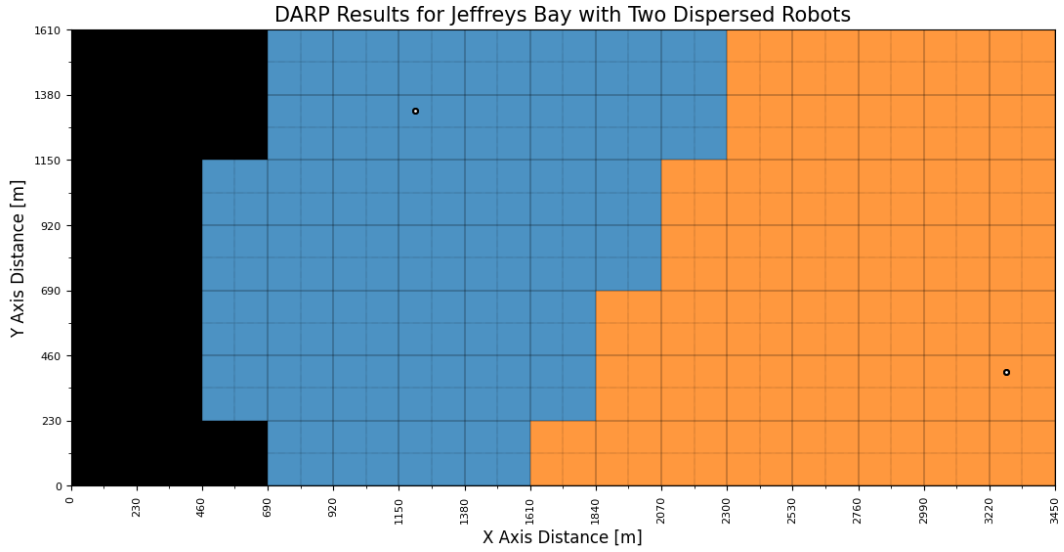
Jeffreys Bay

Figures 6.8a and 6.8b finally show the smallest environment, which naturally only has two UAVs assigned to it. The clustered scenario here does not result in any noticeable changes, but these examples illustrate the algorithm's ability to work with a smaller environment.

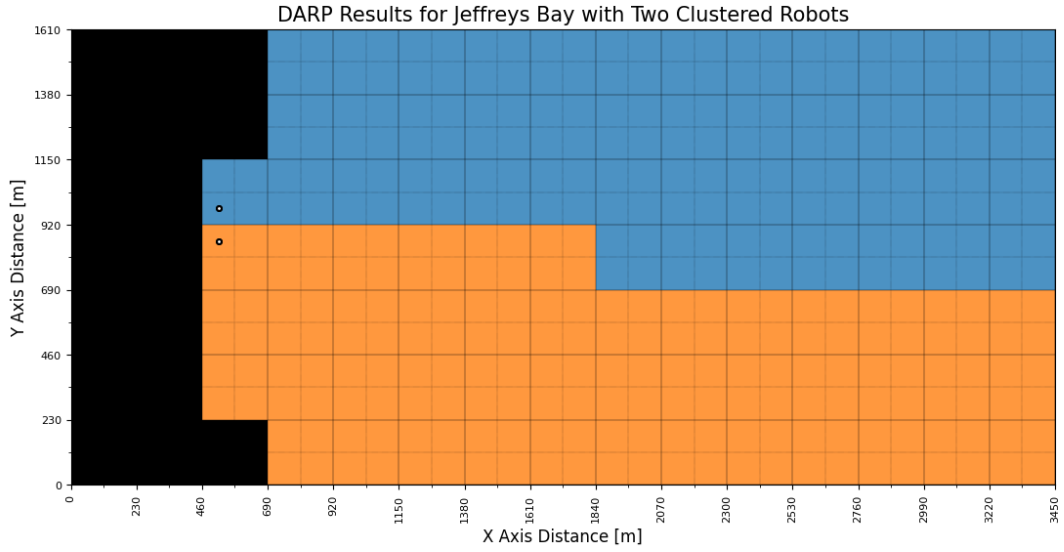
The total environment size for this environment is only 105 cells, with 17 of them being classified as obstacles. The resulting obstacle density is 16%, but the shape of the region of free cells is notably simply. With 88 free cells, a perfect division in this environment would assign 44 cells to each of the two UAVs, and have a resulting discrepancy of zero. In both environments, this perfect division was achieved.

Discussion

The DARP algorithm produced contiguous regions for all the example environments. The marine scenario (Jeffreys Bay) and ground scenario (Aberdeen) produced perfect area divisions, in terms of equal sized sub-regions, as well. The Spitskop environment, which is a mountainous environment with 26% obstacles, also had near perfect area divisions. The DARP algorithm appears well suited to these environments. The area division was sub-optimal for the Champaigne Castle environment, due to its 81% obstacle density and challenging shape. The DARP algorithm is less well suited to this environment. Overall, these illustrative results give preliminary evidence that there may



(a) Dispersed UAVs



(b) Clustered UAVs

Figure 6.8: Results of applying the DARP algorithm to the Jeffreys Bay example environment with two UAVs

be some correlation between obstacle density and how successful the DARP algorithm is.

In each case shown, a different number of UAVs were shown. It is likely in a real world scenario that the number of UAVs would be limited by availability. Depending on the number of UAVs available and the size of the environment, one UAV may need to refuel and fly more than once to cover its region. This project deals with this problem by introducing a ground station for take-off, landing, and refuelling. The UAVs in the environment would have a certain

endurance, and based on this a refuelling plan can be developed. This will be discussed in detail in Chapter 8.

The actual paths of the UAVs will be generated using spanning tree coverage (STC). This is not yet shown in the examples of this section. These paths play an important roll when calculating UAV endurance limitations. There is only a limited path length that a UAV can execute before needing to refuel or recharge, at which point it needs to be back at the ground station to do so.

Chapter 7

Sub-Region Coverage Technique

This chapter discusses the technique used to generate paths for the individual UAVs in the environment. After dividing the environment into sub-regions, a coverage path is devised for each region using spanning tree coverage (STC). Section 7.1 provides an overview of the method employed as well as the motivation for using it. Section 7.2 discusses the generation of a spanning tree for each sub-region. Section 7.3 explains how the spanning tree is circumnavigated to generate the paths for the UAVs. The considerations when using this technique in the SAR context is discussed in Section 7.4. Illustrative examples are then shown in Section 7.5.

7.1 Sub-Region Coverage Overview

In this project, the problem of covering a demarcated search area with multiple UAVs is solved by first dividing the free cells in the environment into sub-regions. Once the environment has been divided into contiguous sub-regions, it is possible to use single robot coverage algorithms to cover each sub-region. If each UAV is guaranteed to achieve coverage of its assigned sub-region, coverage of all the free cells in the environment is achieved.

To avoid backtracking in the search, the environment is represented using large cells. These large cells each represent a set of four smaller cells. These smaller cells are the ones the UAVs are expected to traverse. If a UAV traverses a small cell using one of a few different manoeuvres, this cell is viewed as covered. A path needs to be generated to make sure that the UAV executes one of these manoeuvres in each of the free cells assigned to it. The technique used in this project to achieve coverage is spanning tree coverage (STC).

Figure 7.1 helps to visualize the concept of small cells and large cells. The obstacles in this environment, shown as solid black cells, are the size of a large cell. The smaller cells are indicated on this grid of large cells using dotted lines. This example environment was shown in Chapter 6 with the large cells numbered. Now it shown with actual distances in metres, to root the problem

in the real world.

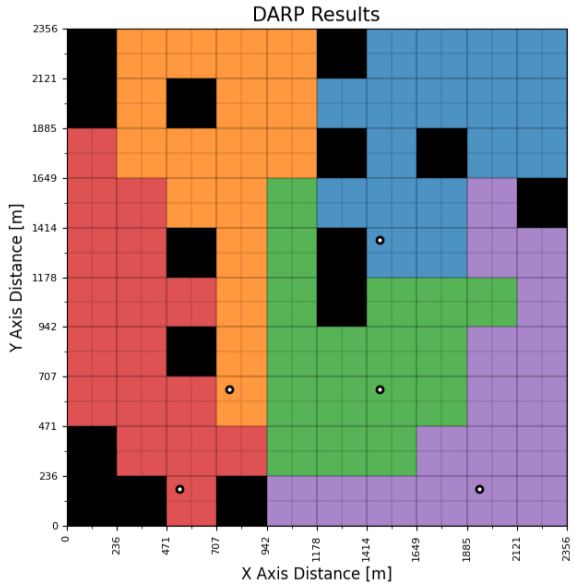


Figure 7.1: Example environment where the divide areas algorithm has been applied, with small cells depicted inside larger cells.

The discretisation sizes are calculated based on the UAV and camera combination used to execute the search plan. In this case the Wingtra I and Sony RX1R II were used.

The starting environment here represents a region of about 5.6 km^2. It is discretised so that the camera is guaranteed to cover a small cell with a straight line manoeuvre or a 90 degree turn. The dynamic constraints of the UAV flying at a constant speed and height are taken into consideration. Any rotation would therefore have a curvature to it. The constant flying altitude of the UAVs is chosen as 210 m above the highest point in the topography. The constant flying speed for the UAVs is chosen as 16 m/s. Based on these decisions, the small cell sizes are determined to guarantee coverage with the specified manoeuvres. These small discrete cells are grouped into the larger cells to form an environment representation that can be used in the divide areas algorithm. This then guarantees that no backtracking would occur when using a spanning tree to achieve coverage.

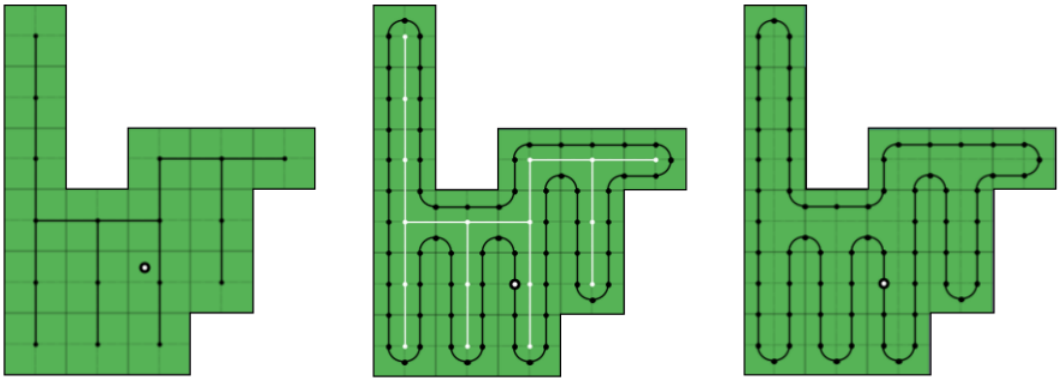
Figure 7.1 shows each sub-region created by the divide areas algorithm in a different colour. One UAV's initial position is within each sub-region. Figure 7.2 shows how a path would be generated for the UAV in a sub-region to cover that region using STC. In particular, the green sub-region is shown here.

Figure 7.2a shows the spanning tree that would be generated for this sub-region. Note how it connects the centres of all the large cells without creating any loops. Figure 7.2b then shows that a path can be generated to circumnavigate this tree. Here, the spanning tree is shown in white, and the

path itself in black. The resulting path covers each cell in the environment using either straight line manoeuvres or 90 degree turns. Complete coverage is therefore achieved.

Figure 7.2c finally shows the path in isolation, without the tree. The path meets the dynamic constraints of the UAVs and guarantees coverage. The UAV initial position, which is shown as the bigger black dot with a white dot at its centre, is where the path both starts and ends. It is therefore a closed loop path.

If this coverage technique is applied to each of the sub-regions, the whole environment will be covered.



(a) Tree (b) Circumnavigation (c) Final path

Figure 7.2: Illustration showing how a spanning tree is used for coverage on one sub-region.

There are a number of advantages to using STC specifically. The most obvious advantage is that it guarantees complete coverage. All the cells will be reached provided the divide areas algorithm creates contiguous sub-regions for each UAV to cover.

Another advantage is the closed-loop path. In this project, a central ground station for take-off and landing will be used when forming a more complete problem with endurance considerations. The UAVs could then take off from this area to reach their initial positions, complete their circuit within their endurance limitations, and then land again after they return to where they started.

The closed loop simplifies the take-off and landing problem. It also means that multiple circuits can be used to formulate a refuelling plan. This will be discussed in Chapter 8 in detail.

The spanning tree algorithm, as it is applied here, means that the UAVs can use simple manoeuvres to cover the area. They only need to be able to execute 90 degree turns and straight line manoeuvres. This means that achiev-

ing complete coverage is easier, since coverage only needs to be guaranteed for these specific motions in each cell.

Ensuring the UAVs can execute their paths within their dynamic constraints is also simplified. If a way is formulated for them to execute each manoeuvre within a cell, then it follows that they can execute their entire paths.

The final benefit of STC is that there is an option to use minimum spanning trees (MSTs). This means that weights can be assigned to certain points of a graph to change the shape of the spanning tree. This can be used to favour certain sweep directions for a specific environment. Certain sweep directions way be more favourable due to weather conditions or simply environment shape.

STC is a grid-based, offline technique. Therefore it has some inherent disadvantages. The resolution of the environment is limited since UAV manoeuvres need to be executed within the confines of the cells. The modularity it offers is very useful in guaranteeing coverage and generating full path plans a priori. However, this discrete environment is only an approximation of the real environment.

The technique used for environment representation in this project introduces a large amount of overlap with the camera FOV when covering a cell. The disadvantage of this is that there is redundant coverage. However, this overlap likely offsets the inaccuracy in the environment representation.

The offline technique means that the algorithm is not adaptable to a dynamic environment. However, there is less need for onboard equipment to plan paths in real-time. Since the UAVs fly very high, a static representation of the environment is realistic. With topographic maps and satellite imagery, it would be fairly easy to formulate a two-dimensional, static environment representation. It is assumed there are no dynamic obstacles to consider for the purpose of this project.

The algorithm also cannot adapt in scenarios where the divide areas algorithm fails, but the divide areas algorithm is quite robust and the scenarios wherein it fails are likely avoidable.

7.2 Spanning Tree Generation

Kocay and Kreher [76] wrote a book in which they address the spanning tree as a structure. Much of the information in this section is with reference to this book. Provided one has a connected and undirected graph, a subset of this graph can be found that is referred to as a spanning tree. A graph constitutes a set of nodes and edges. Figure 7.3 shows an example graph with four nodes and five edges. A spanning tree is a subset of the graph that has a number of properties. It contains all the nodes of the original graph but only enough edges to connect all the nodes to form a connected graph.

The nodes in a spanning tree are connected so as not to form loops. The result is that the number of edges is equal to one less than the number of nodes. A number of possible spanning trees can be generated for the example graph. In this case there are eight possibilities, which are shown in Figure 7.4.

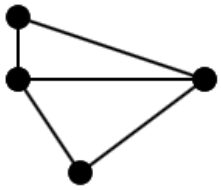


Figure 7.3: Diagram showing an example of a graph with four nodes and five edges.

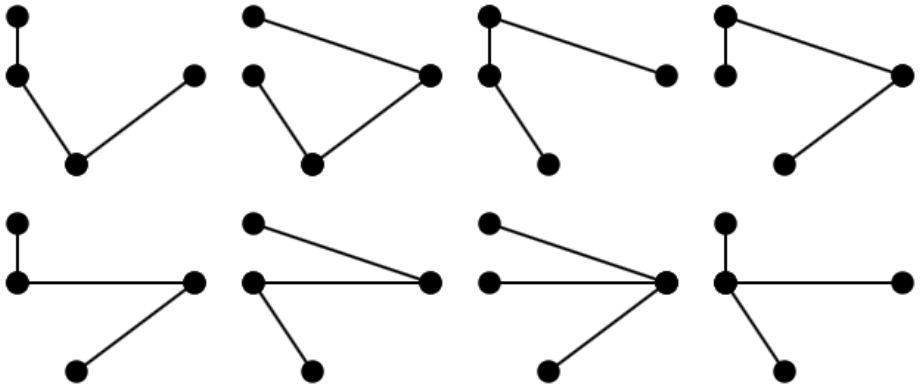


Figure 7.4: Diagram showing possible spanning trees of the example graph.

A specific type of spanning tree seeks to minimise the overall spanning tree weight. This is known as a minimum spanning tree (MST). This is useful for a graph that has weights assigned to its edges. This weight could be representative of a number of different things, for example spatial distance between nodes. A minimum spanning tree would effectively minimise the cost of connecting all the nodes in the tree.

Figure 7.5a shows the spanning tree graph with weights assigned to its edges. There are often multiple MSTs to a graph. In this case there is only one, and it is shown in Figure 7.5b. The total weight of the MST for this graph is 8.

With the grid-based environments used in this project, a graph can be made using the large cells in the environment. Figure 7.6 shows the process of creating a graph using the centres of the large, free cells.

(a) Graph with Weights. (b) Minimum Spanning Tree.

Figure 7.5: Diagrams showing an example of a a graph with weights and the resulting Minimum Spanning Tree of that graph

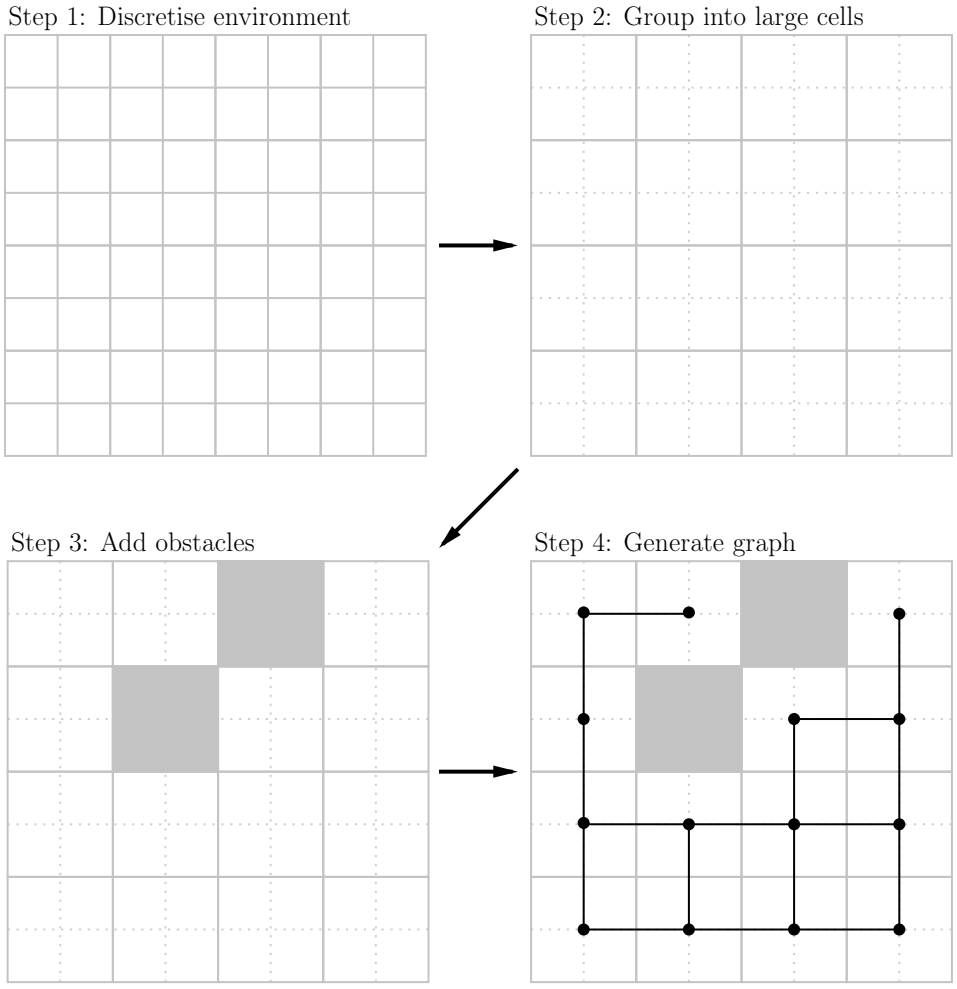


Figure 7.6: Illustration showing how a discrete environment is used to create a connected, undirected graph.

Step 1 in Figure 7.6 shows a square environment that has been discretised into small cells. The number of rows and columns for this environment approximation is required to be even, since they need to be grouped to form the large cells. Step 2 shows how the small cells are grouped into larger cells

for the square environment. Step 3 then shows how obstacles are added to the environment at the resolution of the large cells. In this case, two cells are designated as obstacles.

This environment can now be converted into a connected and undirected graph by introducing a set of nodes and edges. The nodes are at the centres of the large cells that are not obstacles. All the free cells in the environment get connected by the graph.

Step 4 in Figure 7.6 shows the resulting graph in black, with the dots representing the nodes of the graph and the lines between them representing the edges. The nodes are at the corners of small cells, excluding those at the borders of the free region. The edges in turn are along the edges of the small cells and connect the nodes. Altogether, this graph has 14 nodes and 17 edges.

Seeing as the edges provide a path from any one node to any other node in the graph, it is a connected graph. The edges also do not have an associated direction, meaning the graph is undirected. With these properties, the graph can be used to construct a spanning tree.

The spanning tree is still a connected and undirected graph, but if one edge were removed the graph becomes disconnected. Figure 7.7 shows an example of a spanning tree that could be found for this example, with 14 nodes and 13 edges.

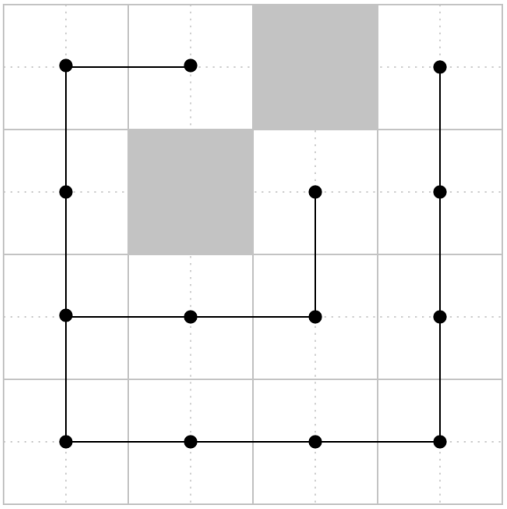


Figure 7.7: Diagram showing a possible spanning tree for the environment graph.

This spanning tree is the first step in achieving coverage for this environment. However, a single spanning tree would be used for a single UAV in that capacity. Therefore, if the divide areas algorithm is applied to generate a sub-region for each UAV, each sub-region would have a spanning tree applied to it as if it were its own environment. Taking the example environment from

Figure 7.1, a set of spanning trees can be found. The result is shown in Figure 7.8. Here the spanning trees are the dark black lines connecting the large cell centres.

The UAV initial positions are adjacent to the spanning trees, at the centres of small cells. The UAVs are expected to traverse the small cells in order to achieve coverage. Therefore, the spanning trees set up a type of barrier for the UAVs to follow in order to cover their respective sub-regions.

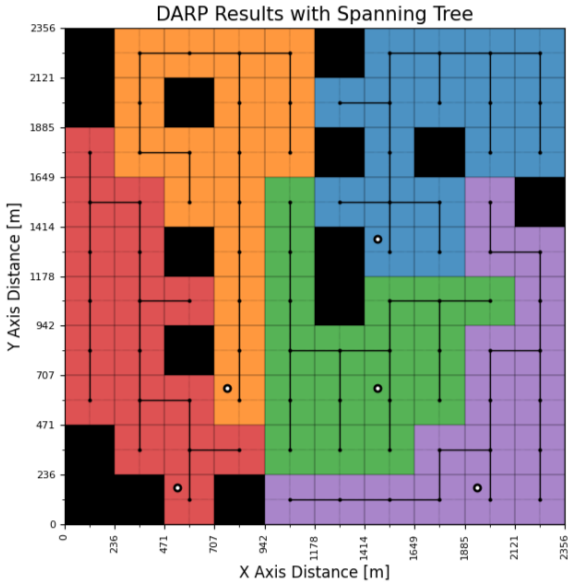


Figure 7.8: Example environment with a spanning tree generated within each sub-region.

The spanning trees in this example are generated using Prim's algorithm [63]. This is an algorithm that generates an MST. For this tree, all edges are weighted equally and any of the possible spanning trees would qualify as an MST. The effect that adding weights has on the coverage paths will be discussed further in Section 7.4.2.

7.3 Path Generation

In order to cover the sub-regions using a spanning tree, the trees are circumnavigated. The clockwise circumnavigation process is described in Section 7.3.1. Dynamic constraints are then taken into account for the generated paths, which is discussed in Section 7.3.2.

7.3.1 Spanning Tree Circumnavigation

Once the spanning tree has been generated, it can be used to achieve coverage of an environment or region. This is done by circumnavigation of the tree and is referred to as offline spanning tree coverage (STC), according to Gabriely and Rimon [53]. When generating a path that circumnavigates the tree, one effectively generates a coverage path that moves through the centres of the small cells, which are the original environment discretisations.

Gabriely and Rimon [53] did not document their process for spanning tree circumnavigation, therefore a unique circumnavigation method was developed as part of this project. Circumnavigation was achieved using a two phase process. Initially, a series of arrows are generated to represent a clockwise motion around the tree. This looks similar to a directed graph, because the edges are assigned directions. However, because these arrows are used to generate a path for tree circumnavigation, there will be two arrows on each edge and the order in which they are used is crucial. In the second phase, these arrows are used to generate the waypoints necessary to circumnavigate the tree.

A convention was established in order to generate the arrows and waypoints consistently. Firstly, once the tree is generated, a starting node is chosen. From this node, a walk is done from one node to the next to form arrows. These arrows are generated in such a way that they can be used to represent a clockwise motion around the tree. Keeping this clockwise convention in mind, waypoints are generated for each arrow.

The direction an arrow points always represents a forward direction within its reference frame. If the next arrow is pointing in the same direction, it is considered a forward motion (F). Three other motions are possible, namely left (L), right (R) and backward (B) motions. It is important to note that, although backtracking is allowed for arrows, this is not the case for the waypoints. A representation of how a reference frame would move with the arrows in the event of a right turn is shown in Figure 7.9.

To generate the arrows, the algorithm first chooses a starting node. This node is one of the nodes on the tree with only one edge. Of the nodes numbered in Figure 7.10a: nodes 0, 10 and 13 would qualify. Assuming node 0 is chosen, the arrows can then be generated as shown in Figure 7.10b. A gap is placed between the arrows here for clarity, but they can be viewed as lying on the edges of the spanning tree.

The arrows follow a clockwise motion, with orange being generated first. Next follows blue and then green. Every time a backward motion is executed, the arrow colours are changed in order to show the order of arrow generation clearly.

In order to generate the arrows in the correct order, a prioritisation strategy is used based on the clockwise convention. When a node has multiple edges, the correct next arrow direction is chosen by cycling through the pos-

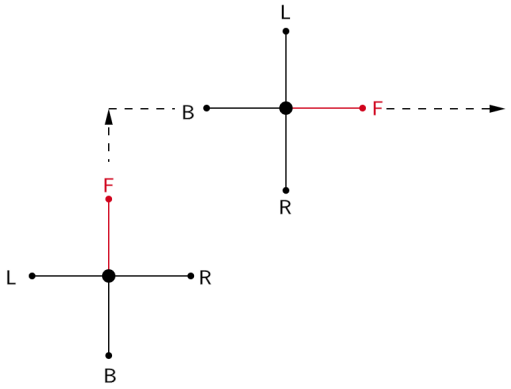
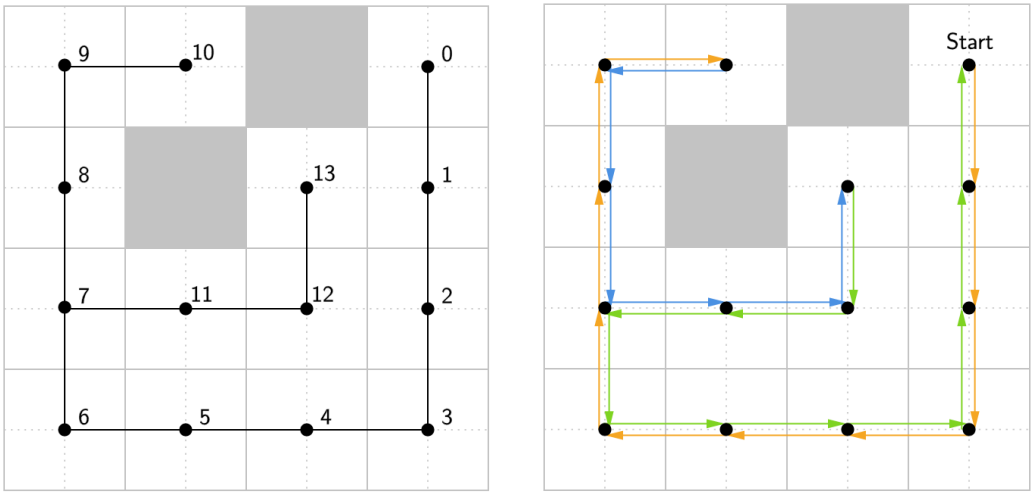


Figure 7.9: Diagram showing how the reference frame representing motions would move with a right turn.



(a) Numbered nodes (b) Arrows

Figure 7.10: Illustration showing how arrows are generated for the first phase of spanning tree circumnavigation.

sible motions in the reference frame (Figure 7.9). Left is prioritised first, then forward, right and lastly backwards.

This convention ultimately results in a clockwise circumnavigation. As an example, observe node 7 with the assumption that the last arrow generated was the orange arrow which runs from node 6 to node 7. The next arrow is chosen by looking at the edges of node 7.

Node 7 does not have a left edge and so the next arrow will not be in that direction. However, it has a forward edge and so that is chosen as the next arrow. The right and backward edges are not considered because of the existence of the forward edge. The next arrow is thus one going from from node 7 to node 8. The order of these arrows is important for waypoint generation.

Figure 7.11 shows the four different motions and how waypoints would be generated for them. The reference frame of the previous arrow is used to establish what motion occurred. For a left turn, one waypoint would be added. Correspondingly, a forward motion would constitute adding two waypoints, a right turn requires adding three, and a backward motion requires adding four waypoints.

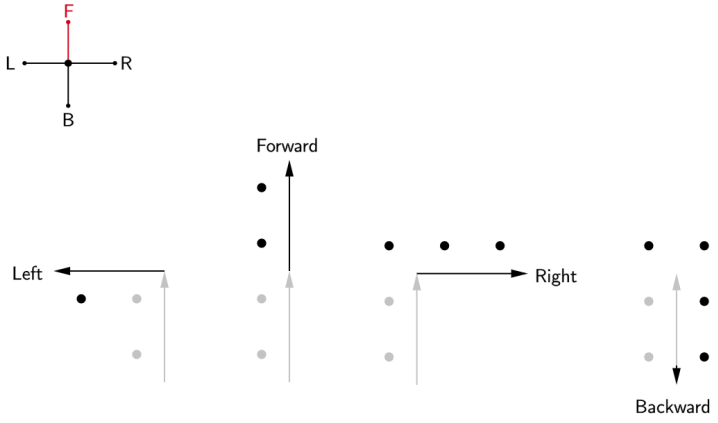


Figure 7.11: Diagram showing the four possible motions of an arrow within a particular reference frame.

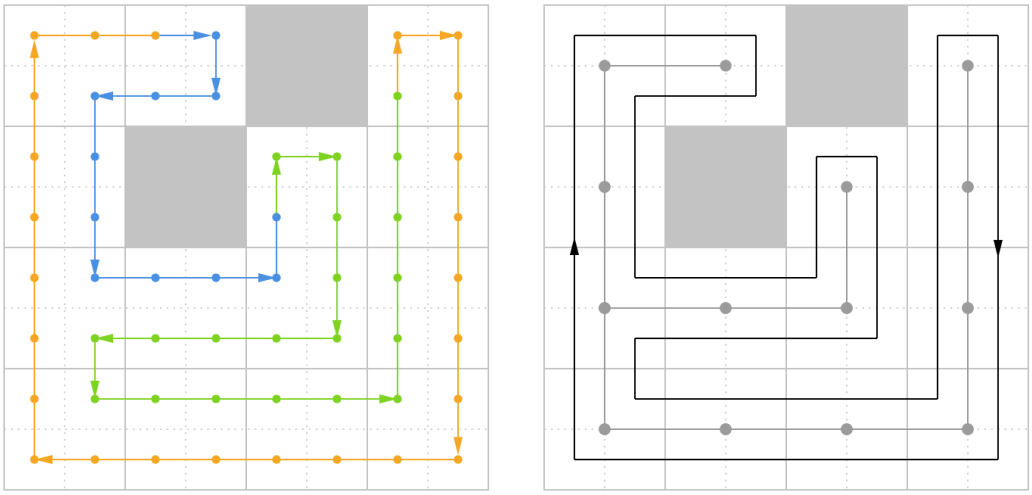
In the figure, the grey arrow represents the previous arrow and the grey dots are its associated waypoints. The reference frame shown is for this arrow. The current arrow is shown in black with the black dots being the waypoints that are generated at this instance.

The start node will always be treated as a backward motion, and so the waypoints phase will always start with the generation of four waypoints. Figure 7.12a shows all the waypoints generated from the arrows of corresponding colours. The first four orange waypoints are linked to the arrow running from node 0 to node 1. All waypoints after that follow the motions as depicted in Figure 7.11. Figure 7.12b shows the path generated from these waypoints overlaid on the original spanning tree, which is shown in grey. This clearly shows the resulting circumnavigation path in black.

The resulting path achieves full coverage of the environment, provided the UAV is capable of orthogonal motions. The assumption here is that the camera footprint will cover the cell when the UAV traverses that cell. The discretisation of the environment into the small cells would be executed with this in mind.

To bring it back to the original problem, this technique should be combined with the divide areas algorithm. This circumnavigation would be applied to each sub-region in a divided environment. Figure 7.13 depicts this on the example environment used in this chapter so far. Here the spanning trees

are depicted in white. The circumnavigation paths with their waypoints are depicted in black.



(a) Path generated by generating waypoints from arrows

(b) Resulting circumnavigation path around spanning tree

Figure 7.12: Illustration of how waypoints are generated from arrows along with the resulting circumnavigation path around the spanning tree.

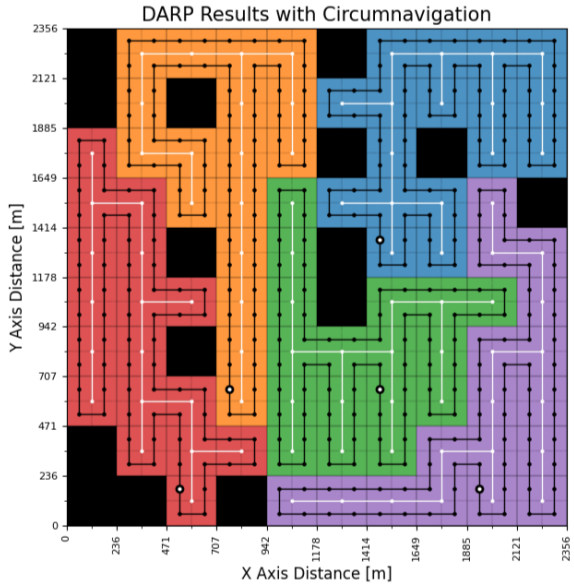


Figure 7.13: Example environment with the spanning tree circumnavigation shown.

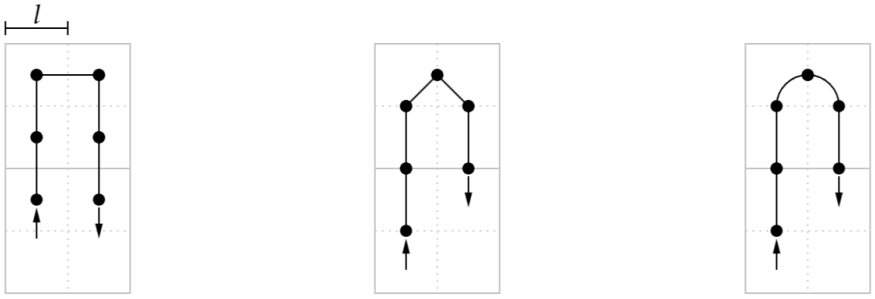
Note how the UAV initial positions lie on one of the waypoints that are generated. This waypoint can be set as the starting point for the UAV paths. The UAVs are each then part of a closed-loop path. If they follow these paths in a clockwise fashion, each UAV would traverse each cell in its assigned sub-region. The ultimate result then is complete coverage of all the free cells, provided the search vehicle is capable of orthogonal movements. The UAV ends where it starts, resulting in the closed loop.

7.3.2 Dynamic Constraints

The circumnavigation of a spanning tree results in complete coverage, but does not yet account for the dynamic constraints of the UAV. The dynamic constraints should be taken into account when discretising the environment to ensure coverage. It is assumed that the motions used here would still result in complete coverage as a UAV traverses a cell.

The simple motions of the circumnavigation mean that dynamic constraints only come into play for the 90 degree turns. To facilitate the process of adding them, the waypoints that are generated in the circumnavigation are all shifted by half a cell. This places diagonal lines on each cell where a turn occurs. These diagonal lines can then be replaced with a circular waypoint.

The result is a set of waypoints for the UAV that are either straight line waypoints or circular waypoints. To illustrate this, the waypoints that are generated for a backward motion are shown again in Figure 7.14a. The arrows here show the direction of motion of the UAV, which is clockwise around the spanning tree.



(a) Original waypoints (b) Half shifts (c) Circular waypoints

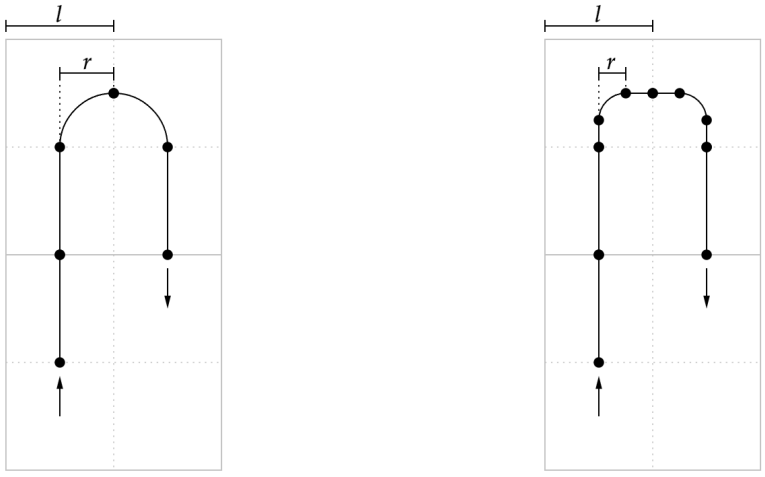
Figure 7.14: Diagrams showing how the original waypoints are shifted to form a set of circular and straight line waypoints.

The dimension of a small cell (l) is indicated in the figure. The spanning tree is not shown, but it would have a vertex spanning from the centre of the bottom large cell to the centre of the top large cell.

The result of applying the the half shifts to the waypoints is shown in Figure 7.14b. The shifts are done counter clockwise, in the direction opposite to the direction of motion. The two 90 degree turns become diagonal lines which can now by converted into circular waypoints. Figure 7.14c shows the curves that replace the diagonal lines. If these curves were given as input to a UAV, it would be represented as a radius and an angle, instead of the point coordinates that represent the straight lines.

Equation 3.7 showed the calculation to determine the minimum turning radius of a UAV for a limiting bank angle and speed. It is assumed that the bank angle chosen can be sustained throughout the path and that it does not result in an very costly manoeuvre in terms of energy consumption. The speed of the vehicle and a sustainable bank angle is chosen as part of the environment discretisation process.

This minimum turning radius will be used as the turning radius of the UAV for all 90 degree manoeuvres. It is possible that this radius is equal to the size of half the discretisation ($r = \frac{l}{2}$). This is depicted in Figure 7.15a. In this case, the diagonal line from the half shifts would be replaced by one circular waypoint. There is also the case where the radius is smaller than this dimension ($r < \frac{l}{2}$). In this case, two straight line segments are added to ensure the waypoints still form a closed loop. This is depicted in Figure 7.15b. The discretisations are designed explicitly so that the radius would never exceed this dimension.



(a) $r = \frac{l}{2}$ (b) $r < \frac{l}{2}$

Figure 7.15: Diagrams showing how the waypoints look when the turning radius is equal to or smaller than half the discretisation size.

It is also possible to simply increase the turning radius of the UAV to match the dimension of half the square discretisation. For this project, the minimum turning radius is simply used as the turning radius.

Dynamic constraints can be added to the circumnavigation paths in the example shown in Figure 7.13. In doing so the UAV initial positions would also get shifted by half a cell. Figure 7.16 depicts just the intermediate step of the counter clockwise half-shifts of the paths. The spanning tree is no longer depicted in this figure.

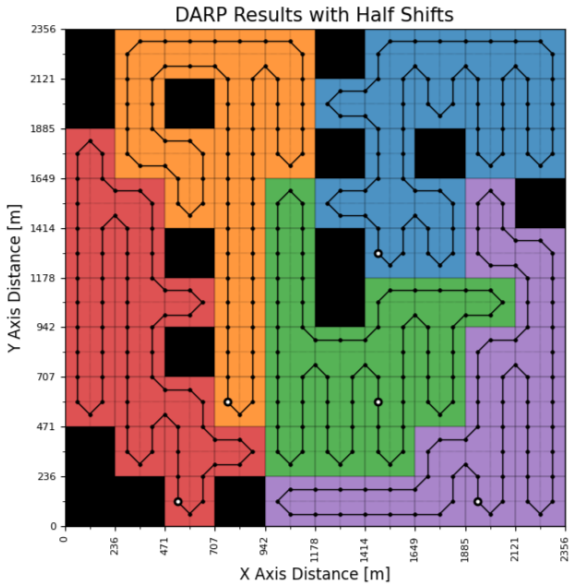


Figure 7.16: Example environment with waypoint half-shifts shown.

Figure 7.17 goes on to show the diagonals being replaced by circular waypoints. This figure represents the final paths of the UAVs to cover their assigned sub-regions.

For this example, the minimum turning radius was calculated as roughly 56 m. Half the size of the square discretisation was 58.9 m. The turning radius is smaller than half the square discretisation ($r < \frac{l}{2}$) so the scenario in Figure 7.15b applies. The waypoints are very close together for the straight line segments, though, so they are not really distinguishable in the figure.

It is possible to maintain the initial positions of the UAVs and not shift them with the waypoints. However, in this application, the initial positions of the UAVs are chosen by the search team. After take-off, the UAVs would navigate to their assigned initial positions to execute their search. Therefore, maintaining these initial positions exactly is not critical.

The shifts also prove useful in ascertaining the initial trajectory of the UAVs. This trajectory is important to know for a fixed-wing UAV. A means can then be found for the fixed wing to take off and reach the required altitude, position, and heading to start the search of its region.

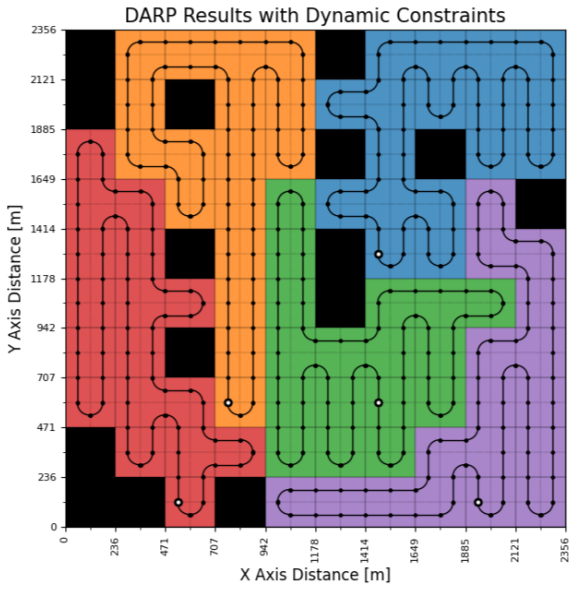


Figure 7.17: Example environment with dynamic constraints shown.

7.4 Spanning Tree Coverage for SAR

This section discusses three prominent considerations when applying STC to a SAR scenario. Section 7.4.1 discusses survivor detection in a SAR operation. Section 7.4.2 goes on to mention how weather conditions can be accounted for using spanning tree weights. Lastly, UAV endurance limitations are addressed in Section 7.4.3.

7.4.1 Survivor Detection

The main objective of a SAR operation is to locate a survivor or group of survivors. Survivor detection would occur in real-time. A camera onboard a UAV would locate a target and then follow some procedure to inform the relevant parties of the target location. For the purpose of this project it is assumed that if a target is present within the bounds of a discrete element, it would be detected by the UAV as it traverses that small cell.

There is by definition camera overlap between the cells. The overlap regions are not considered for target detection, though. The portion of the camera FOV that is within the cell the UAV is traversing, is assumed to be where a target can be spotted.

This algorithm does not address what happens once the target is detected. The UAVs may, for example, plan paths to land directly after detecting the survivors, or they might continue on their coverage paths.

Regardless of what happens at detection, it is possible to calculate the time to locate the survivor(s). It is important, first of all, to reiterate the

assumption that the target is assumed to be stationary for the UAV search duration.

Figure 7.18 shows a potential target location within the example environment used previously. It is shown as a black **X** mark. This is a simulated target position. In practice, the target location is unknown until one of the UAVs enter that cell.

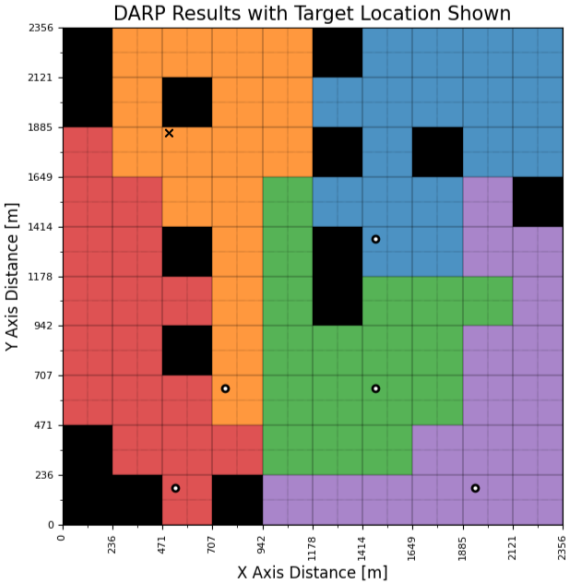


Figure 7.18: Example environment showing the DARP division and target location.

To simulate a detection event, a snapshot can be taken of the UAVs when the target is found. If a UAV enters the cell the target is in, it completes its manoeuvre in that cell. Similarly, every other UAV's path can be drawn to the last waypoint prior to the detection event.

The resulting snapshot for this scenario, with the paths taken from Figure 7.17 can be seen in Figure 7.19. Because these paths are executed at a constant speed (V_f), time is simply a question of determining distance.

The distance of a straight line waypoint would always be the length of the discretisation itself. Therefore, with a constant flying speed, the time associated with a straight line waypoint can be calculated using

$$t_s = \frac{l}{V_f} \tag{7.1}$$

where t_s is the time to execute a straight line manoeuvre, l is the dimension of a small cell discretisation and V_f is the constant flying speed of the UAVs.

A circular waypoint length may be the length of two short straight line segments and the arc length of the 90 degree curve. If the radius of the curve

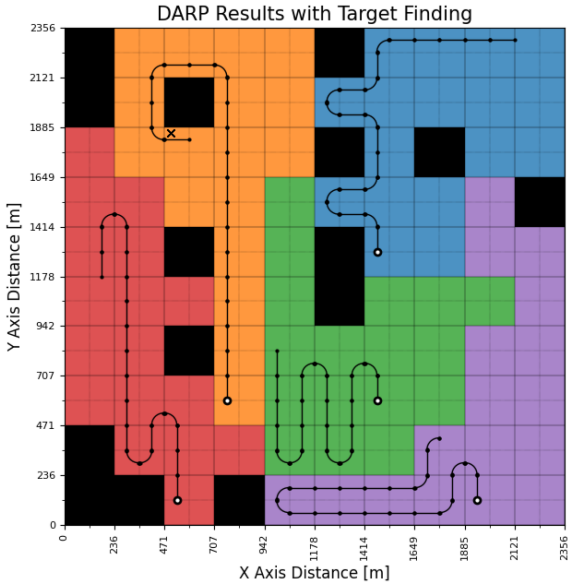


Figure 7.19: Snapshot of example environment at time of target detection.

is equal to half the dimension of the square cell, it becomes just an arc length. The calculation for the time associated with a 90 degree turn is given by

$$t_c = \frac{2 \cdot (\frac{l}{2} - r_{\min}) + r_{\min} \cdot \frac{\pi}{2}}{V_f} \tag{7.2}$$

where t_c is the time to execute a 90 degree turn, $r_{\min}$ is the minimum turning radius of the UAV and V_f is the constant flying speed of the UAVs.

For this example, when making use of these time equations, the time at which the target is found in the search is two-and-a-half minutes into the search. If the total path lengths for these sub-regions were executed, they would take between seven-and-a-half and eight minutes each.

7.4.2 Weather Conditions

Weather conditions are regularly a consideration in SAR operations. For target detection there may be visibility consideration when there is heavy rain or fog. High winds or wet conditions may also make it impossible for UAVs to assist.

The advantage of using UAVs is that no human life is put at risk during the search. Therefore, even if there is a high risk of losing a UAV during a SAR operation, no lives are put at risk.

There is one way in which the spanning tree style coverage algorithm can account for weather conditions, and this is using the weights of an MST. The algorithm used to generate the spanning trees in this section is called Prim's algorithm. It is an MST algorithm, that seeks to minimise the weight of a spanning tree.

Up until this point, the weights of all the tree edges were made equal. By looking at the results of Figure 7.17, one can see that the way the algorithm is set up automatically favours an up-and-down type sweep. This can be altered to accommodate weather conditions.

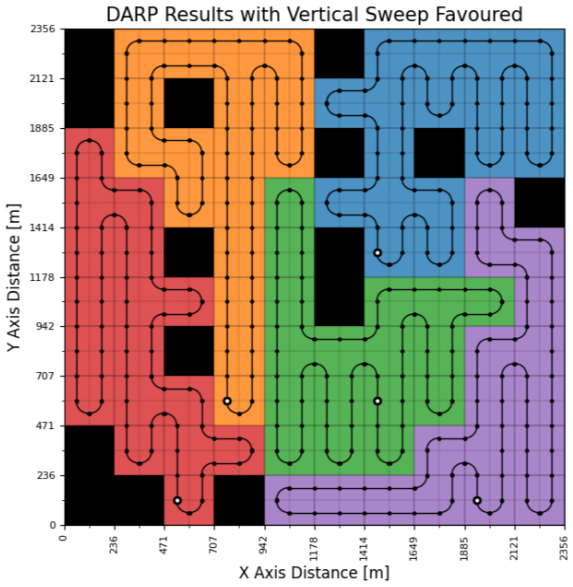


Figure 7.20: Example environment where the y-dimension is favoured during path generation.

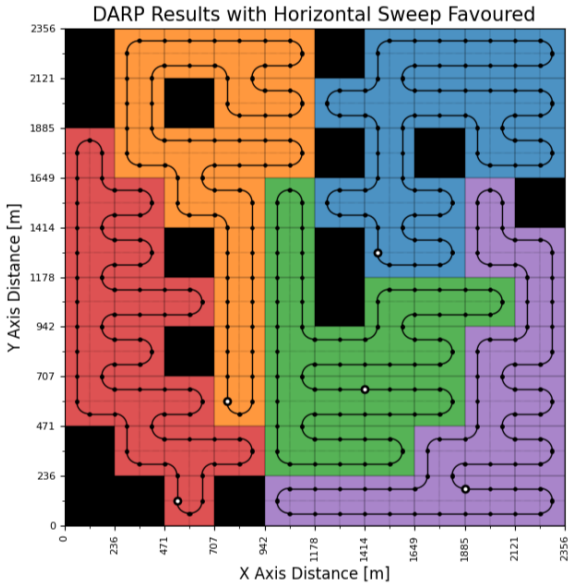


Figure 7.21: Example environment where the x-dimension is favoured during path generation.

A crosswind may make it more challenging for UAVs to adhere to their trajectories. Therefore it is advantageous to favour flying in the direction that is on the same axis as the wind direction, or the axis on which the largest component of the wind is.

The spanning tree for the grid-based environments of this project has two axes that can be favoured. If the y-axis is favoured, the resulting paths will have paths that closely resemble a back-and-forth sweep along this axis. This is shown in Figure 7.20. Here, the edges that are parallel to the x-axis would have higher weights than those on on the y-axis. Similarly, the weights can be altered to favour a sweep along the x-axis, as shown in Figure 7.21. Here the edges that are parallel to the y-axis would have higher weights. Due to the closed-loop nature of this algorithm, an axis can be favoured, but a direction cannot. Therefore, if the wind is on an axis, the UAV favours upwind and downwind equally, while avoiding a crosswind.

7.4.3 UAV Endurance

Once a type of UAV is chosen for an application, it is important to know its endurance capabilities. Longer endurance means that a UAV can continuously search a larger region without refuelling or recharging. In general, longer endurance means less UAVs would be necessary to cover a region. If refuelling is factored in, longer endurance means the search time would not be prolonged as much by refuelling events. There would be less or no refuelling events.

Often, solar-powered and light-weight vehicles have longer endurance [77]. Solar panels and light-weight materials may however be more expensive. Flying in overcast or harsher conditions may also prove challenging. It would be up to the search team to make an informed decision for a specific scenario regarding which UAV is most appropriate.

The search team must select a UAV that suits their robustness requirements while also having a reasonably good endurance. They may also need to factor in likelihood of UAV loss. It may for example be a good decision to use more readily available and cheaper UAVs in a scenario with high risk of UAV loss.

A number of implementations seek to minimise rotations in the UAV paths [49, 52]. This is because drag is increased during a rotation as opposed to flat flying. This means that to maintain a constant speed, thrust needs to be increased, and so energy consumption is increased [78].

Depending on the shape of a sub-region it may be beneficial to sweep along one dimension, so as to reduce the number of rotations in a path. However, it is important to consider that the distance flown in a cell is longer when executing straight line motions than 90 degree rotations, provided the turning radius is non-zero. The benefit of minimising rotations may therefore be diminished by the difference in flight path lengths for the different manoeuvres.

The way endurance is treated in this project is based on the predicted flight times of a particular UAV. This prediction is used to calculate the number of cells that can be assigned to that UAV for one flight. This is then compared to the total number of cells in the environment. In this way a minimum number of flights by this UAV is calculated. If this number of flights is more than the number of UAVs available, refuelling or recharging may be necessary to achieve complete coverage.

To show this mathematically, one can start by assuming a specific number of UAVs are available, represented by $(n_r)_{\text{avail}}$. Based on the type of UAV used, a predicted flight time of that UAV (T_p) can be found in its specifications. To be conservative, it is assumed that this time is representative of flight time when flying level and straight. Therefore, rotations and changes in altitude would constitute reducing this time.

For this project, a 30% safety factor ($k_t = 1.3$) will be applied to any rotation, as well as to any take-off and landing time, when representing energy consumption. Because of this, the time to execute a flight path is differentiated from the energy consumption, which is also measured in time.

The time taken to execute the flight path would be a matter of adding the time taken to execute the straight line manoeuvres and the time to execute the 90 degree turns in a path. This is done using

$$T_f = n_s t_s + n_c t_c \tag{7.3}$$

where T_f is the total time to execute a flight path. n_s is the number of straight line manoeuvres and n_c is the number of 90 degree turns in the path. t_s and t_c are the times to execute a straight-line manoeuvre and a 90 degree turn respectively. .

The energy consumption during flight is also measured in units of time, but the safety factor is applied to rotations. This is calculated using

$$(T_f)_e = n_s t_s + k_t(n_c t_c) \tag{7.4}$$

where $(T_f)_e$ is the total energy consumption during flight and k_t is a positive safety factor. Any additional components of flight, such as take-off and landing times, are not shown in these equations. This would have to be added to measure whether the flight paths are within the energy limits if the UAV.

To ensure the UAVs complete their paths within their energy limits, the predicted flight time can be used to calculate whether any refuels would be necessary. The time to execute a 90 degree turn (t_c), with the safety factor applied, can be compared with the time taken to execute a straight line manoeuvre (t_s). The maximum of these is then used to conservatively represent the energy taken to cover a specific cell. The maximum energy to cover a large cell is the four times this amount.

The total available flight time, which represents the total available energy, can be divided by this to get a maximum number of large cells that can be

assigned to a specific UAV, as given by

$$\text{cells}_{\max} = \frac{T_p - k_t T_m}{\max(k_t t_c,\ t_s) \cdot 4} \tag{7.5}$$

where $\text{cells}_{\max}$ is the maximum number of large cells that can be assigned to a specific UAV, T_p is the predicted flight time of the UAV and T_m is an estimation of the time to execute any manoeuvres not included in the coverage path of a UAV, such as take-off and landing times, which also have the safety factor applied to them.

The suggested maximum number of cells per UAV can be used to calculate the required number of UAVs to cover the environment without needing refuels. It can be calculated by dividing the total free cells in the environment with the maximum number of cells that can be assigned per UAV. This value is referred to as the equivalent number of UAVs. It represents the number of UAV initial positions required to search the area. Several UAV initial positions would then be assigned to one UAV, depending on how many UAVs are physically available. Assuming all the UAVs refuel an equal number of times, the equivalent UAVs are adjusted to be a multiple of the number of available UAVs in the second step of the equation given below as

$$\begin{aligned} (n_r)_{\text{eq}} &= \frac{\text{cells}_{\text{free}}}{\text{cells}_{\max}} \\ (n_r)_{\text{eq}} &= \text{ceil}\left(\frac{(n_r)_{\text{eq}}}{(n_r)_{\text{avail}}}\right) \cdot (n_r)_{\text{avail}} \end{aligned} \tag{7.6}$$

where $\text{cells}_{\text{free}}$ are the total number of free cells in the demarcated search area, $\text{cells}_{\max}$ is the maximum number of cells that a UAV can cover in a single flight before having to refuel, $(n_r)_{\text{eq}}$ is the equivalent number of UAVs, and $(n_r)_{\text{avail}}$ is the available number of UAVs.

The number of refuels can then easily be represented as one less than the equivalent number divided by the available UAVs. The equivalent UAVs ultimately represent the number of UAV initial positions, which are a multiple of the number of available UAVs. Note that refuels, for the purpose of this project, will refer to any act of getting the UAV flight ready again. This may be a refuel, recharge, or even a battery change. The calculation for getting the number of refuels is given by

$$\text{refuels} = \frac{(n_r)_{\text{eq}}}{(n_r)_{\text{avail}}} - 1 \tag{7.7}$$

where $(n_r)_{\text{avail}}$ is the number of available UAVs and $(n_r)_{\text{eq}}$ is the equivalent number of UAVs, which equal the number of sub-regions.

7.5 Illustrative Examples with Different Environments

This section shows examples of the coverage paths that were generated for the real-world environments that were introduced and discretised in Chapter 5, and divided into sub-regions in Chapter 6. Two of the four real-world environments are shown with the paths generated using STC laid over the sub-regions generated by the divide areas algorithm. The other two environments are not shown, since this would not add any new information. How their coverage paths would look is fairly intuitive.

Spitskop, Dispersed Initial Positions

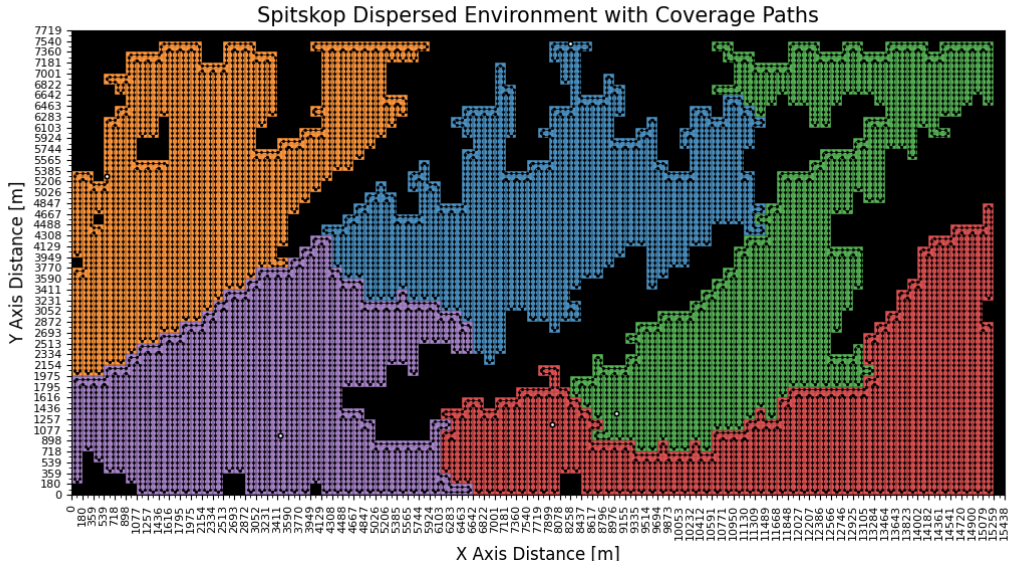
Figure 7.22 shows the Spitskop example environment with dispersed UAV initial positions. Figure 7.22a shows the full paths generated by the spanning tree algorithm, with dynamic constraints incorporated. Figure 7.22b shows a snapshot of this same environment at the time of target detection.

The type of UAV used to cover this environment is the Strix 400. According to its specifications, this UAV can fly for between 9 and 10 continuous hours. This endurance can be attributed to its use of both solar and battery power [73].

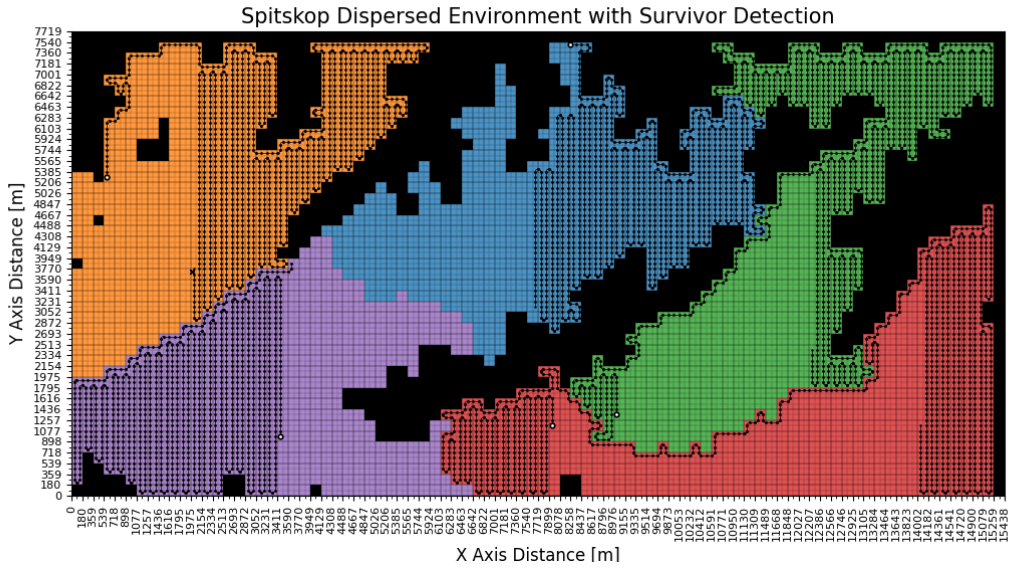
However, the actual path lengths for this area division with five UAVs is roughly 190 km per UAV. The selected constant speed chosen for the flights is 14 m/s, which is the cruise speed for this UAV. The time to complete these flight paths is therefore under 4 hours. This is well under the endurance limit.

The energy consumption during flight, should however have a safety factor applied so that it can be compared to the endurance limit. Therefore, Equation 7.4 is used to calculate the times representative of energy consumption. The values are also just under 4 hours, and so are still within the endurance limit, meaning that it is feasible to cover this entire environment with the available UAVs, without needing to recharge them. When adding the safety factor for rotational movements, the times increase only by around 7 minutes.

The flight metrics for each UAV is shown in Figure 7.23. This is a table that the algorithm outputs upon completion. The first column is the UAV index. For each UAV, a series of time values are shown in minutes. From left to right, these values represent flight path times (T_f), energy consumption times ($(T_f)_e$), endurance limit (T_p), and the difference between the endurance limit and the energy consumption ($T_p - (T_f)_e$). The time difference is representative of the remaining energy available for landing, take-off, or any other additional manoeuvres since these values are not included in the energy consumption calculation. The distance value shows the flight path lengths, and then the final column shows the number of rotations that is in each flight path.



(a) STC paths.



(b) Survivor detection snapshot.

Figure 7.22: Dispersed Spitskop example environment with the coverage paths and survivor detection.

Robot	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]	Distance [km]	Rotations
0	227.2	234.4	540.0	305.6	190.9	280
1	227.8	233.3	540.0	306.7	191.4	214
2	226.8	234.9	540.0	305.1	190.5	318
3	228.0	235.2	540.0	304.8	191.5	284
4	228.1	234.2	540.0	305.8	191.6	242

Figure 7.23: Table generated by the program for the dispersed Spitskop example environment.

Since the Time Excess column has positive values, these paths are feasible for these UAVs without needing to recharge their batteries. It is important to account for landing and take-off before making this deduction though. This will be addressed in Chapter 8. The time limit used is conservatively chosen as 9 hours here, since this is the lower end of the predicted endurance.

The flight metrics show that the flight paths are feasible, but an important thing to note is the time it takes for this coverage plan to locate a survivor in the environment. A target is introduced at a random location. For this environment, the target is placed at a distance of 2 km on the x-axis and 3.7 km on the y-axis. The time of detection with these coverage paths is roughly 2 hours into the search. The target is shown as an "**X**" and can be found in the orange sub-region.

Spitskop, Clustered Initial Positions

Figure 7.24a shows the STC paths generated for the Spitskop example environment, but with the clustered UAV initial positions. Figure 7.24b shows the survivor detection using these paths.

The target location is the same as with the dispersed example, but in this case it appears in the red sub-region. However, due to the different shapes of the sub-regions, the time to find the target is roughly 3 hours and 5 minutes. It therefore takes longer for the UAVs to find this target for the clustered example than the dispersed example. This is an intuitive result since the UAVs are positioned further from the target in the clustered example.

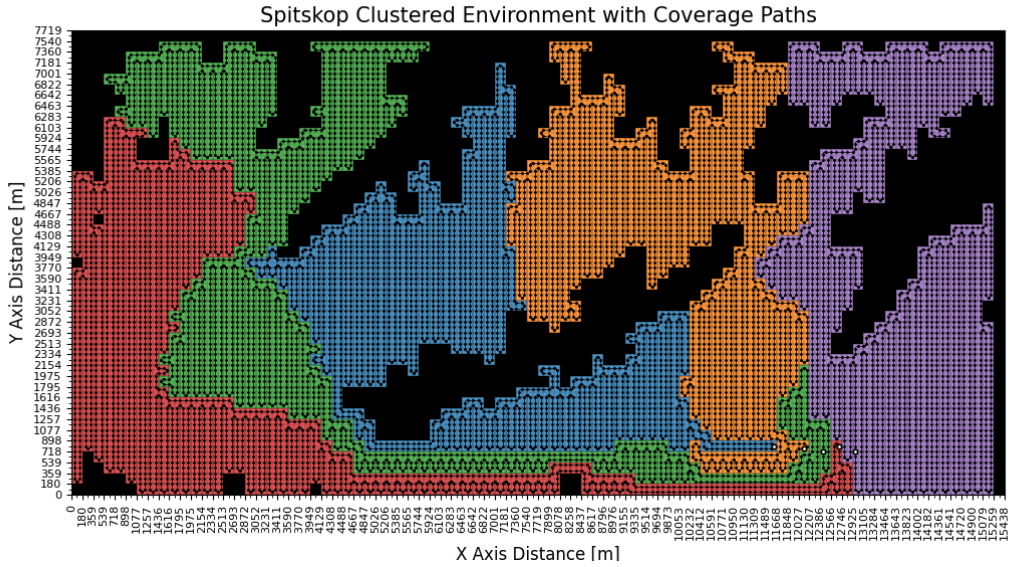
Figure 7.25 once again shows the table generated by the algorithm. Overall the time and distance values are very similar to the dispersed example. This means that the DARP algorithm is very successful in dividing the areas equally amongst the UAVs for both versions of the example.

If, for the moment, we assumed that there are no additional manoeuvres such as take-off and landing times, one can calculate how many UAVs are needed to cover this environment without refuels. Equations 7.5 and 7.6 can be used to do this. If the value is rounded up to the nearest integer, the number of required UAVs is three. This would be the case for both the clustered and the dispersed scenario.

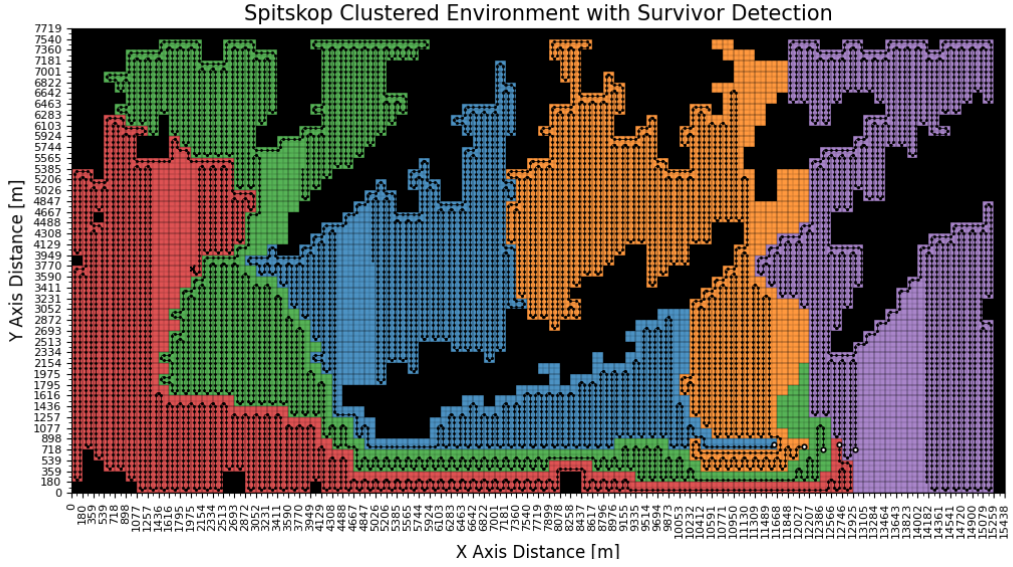
Aberdeen, Clustered Initial Positions

The next examples discuss the Aberdeen example environment. In this case, only the clustered initial positions example is shown. It is however, first shown with the up-and-down sweep favoured and then a left-to-right sweep.

These examples make use of the Wingtra I. This UAV has a significantly lower endurance than the Strix 400 so these examples show a scenario wherein refuelling would be beneficial. According to the specifications, the Wingtra



(a) STC paths.



(b) Survivor detection snapshot.

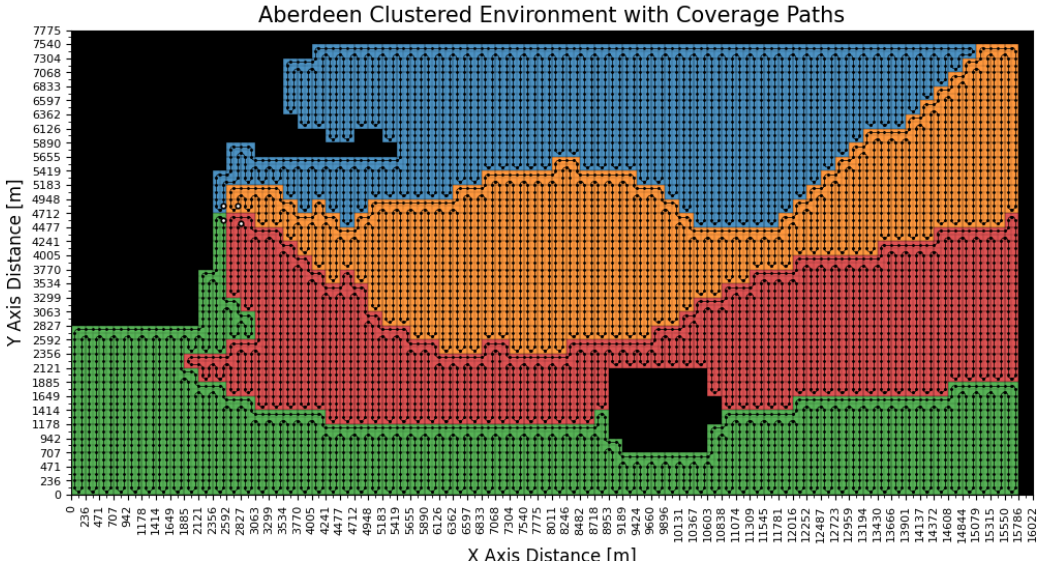
Figure 7.24: Clustered Spitskop example environment with the coverage paths and survivor detection.

Robot	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]	Distance [km]	Rotations
0	226.3	234.5	540.0	305.5	190.1	322
1	226.4	234.5	540.0	305.5	190.2	318
2	223.9	234.9	540.0	305.1	188.0	434
3	227.1	234.9	540.0	305.1	190.7	308
4	227.8	234.3	540.0	305.7	191.3	256

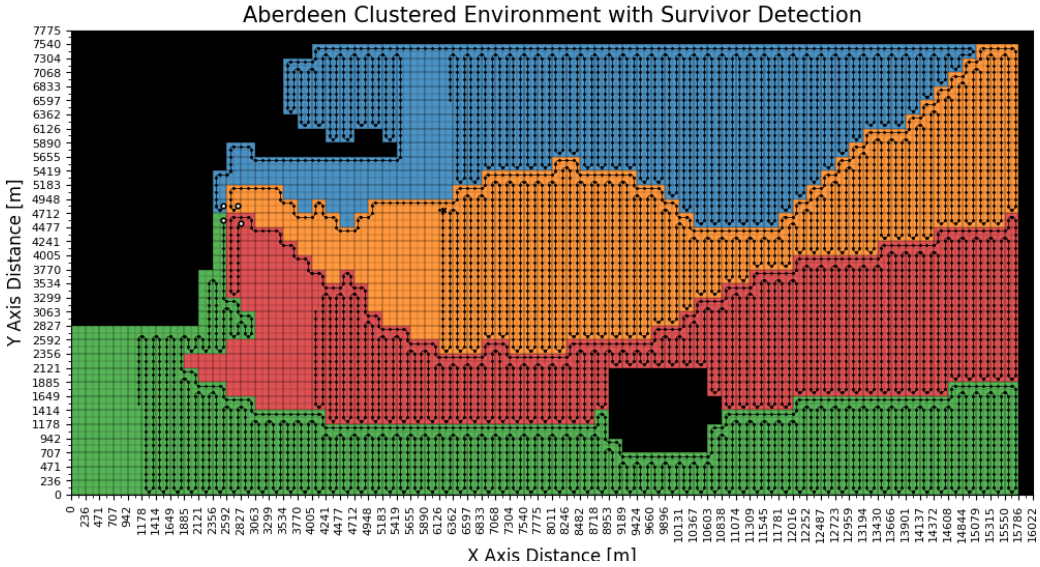
Figure 7.25: Table generated by the program for the clustered Spitskop example environment.

I has a maximum endurance of between 42 and 59 minutes [67]. The lower estimation of 42 minutes is used in these examples.

Figure 7.26a shows the paths generated when four clustered UAVs are used in this environment. Here the paths favour a up-and-down sweep. A target is placed at a distance of 6.2 km on the x-axis and 4.7 km on the y-axis, which falls in the orange sub-region. The detection of this target is depicted in Figure 7.26b.



(a) STC paths.



(b) Survivor detection snapshot.

Figure 7.26: Clustered Aberdeen example environment with the coverage paths and survivor detection.

The time to detect the target in this case is 3 hours and about 26 minutes. This is assuming these paths can be executed of course. In reality, this implementation would require refuelling, which would naturally alter the time it takes to find the target. Once again it is also important to factor in take-off and landing times in a real world scenario.

The table generated by the algorithm is shown in Figure 7.27. The actual flight times (T_f) are about 3 hours and 40 minutes. When factoring in energy consumption this time (T_e) increases by roughly 8 minutes in each case. This value increases slightly with the number of rotations in a path. The energy consumption is far more than the limiting time of 42 minutes. Therefore, the Time Excess column has negative values, which reinforces that refuelling would be necessary for this case.

Robot	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]	Distance [km]	Rotations
0	220.5	227.9	42.0	-185.9	211.7	252
1	220.3	228.5	42.0	-186.5	211.5	280
2	219.3	228.6	42.0	-186.6	210.5	320
3	219.5	228.0	42.0	-186.0	210.7	292

Figure 7.27: Table generated by the program for the clustered Aberdeen example environment.

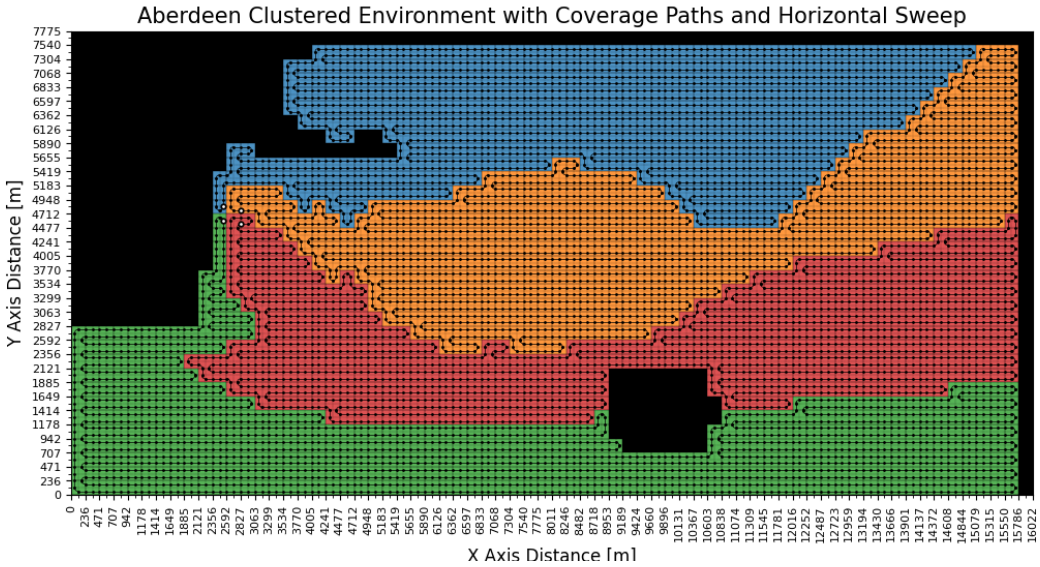
As what was done with the Spitskop environment, one can calculate the number of UAVs that are needed to cover this environment without refuelling. This is once again assuming that there are no additional manoeuvres such as take-off and landing. According to Equations 7.5 and 7.6, the number of UAVs required would be 23. This is an unrealistic amount to have available, therefore reinforcing the need for a refuelling protocol.

Figures 7.28a and 7.28b show the same Aberdeen example but with a left-to-right sweep favoured. Because of how the paths are oriented in this case, the target is found in 30 minutes.

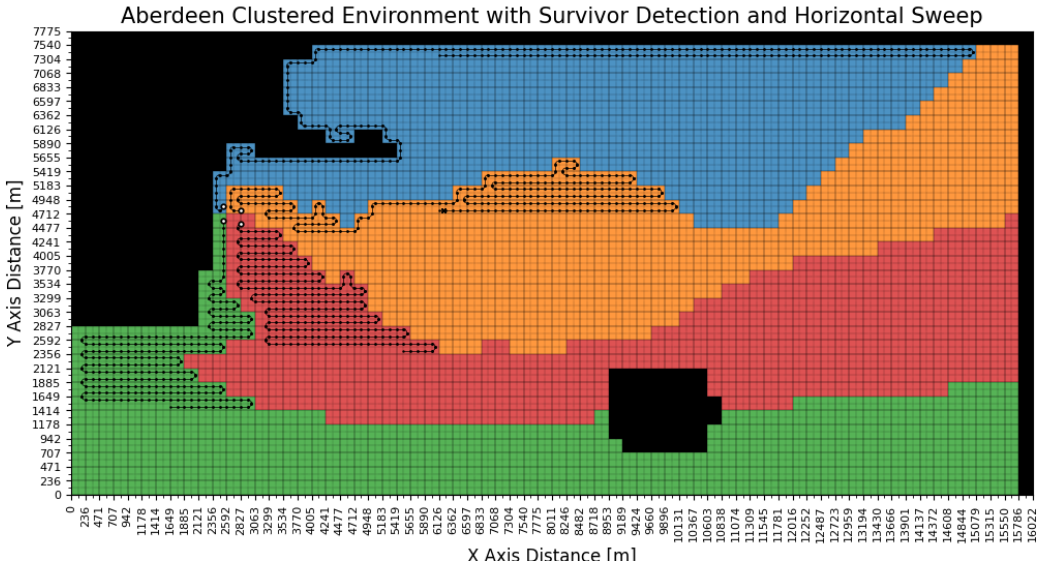
According to the table in Figure 7.29, the distances and flight times for this case are quite similar to the up-and-down sweep. The longest flight path here is 215 km long, which is only slightly longer than the longest path for the up-and-down sweep, which is 211.7 km.

Although the flight paths are slightly longer, the number of rotations for this example is significantly reduced. This is because the sub-regions are longer in the favoured dimension in the sweep. For the up-and-down sweep, there were up to 320 rotations in one path. For this left-to-right sweep, the maximum rotations in one path is 188.

This difference in rotations means that the energy consumption values do not increase by as much in this case. This is anecdotal evidence that reducing rotations could decrease energy consumption. However, the value only changes by a few minutes. Overall, the Time Excess value remains similar for both scenarios. There is no immediate gain in minimising rotations unless they use



(a) STC paths.



(b) Survivor detection snapshot.

Figure 7.28: Clustered Aberdeen example environment with the coverage paths and survivor detection when using a left-to-right sweep.

Robot	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]	Distance [km]	Rotations
0	223.9	227.3	42.0	-185.3	215.0	114
1	222.6	228.1	42.0	-186.1	213.7	188
2	224.8	227.7	42.0	-185.7	215.8	98
3	222.8	227.5	42.0	-185.5	213.9	158

Figure 7.29: Table generated by the program for the clustered Aberdeen example environment with a left-to-right sweep.

significantly more energy than flying straight. In this case, the assumption was that rotations use 30% more energy than flying straight.

Chapter 8

Central Deployment and Flight Scheduling

To bring the implementation in this project closer to a real-world SAR operation, this chapter details the inclusion of central deployment as well as flight scheduling and UAV endurance considerations. Section 8.1 gives an overview of the methods used. Section 8.2 looks at the procedures used to calculate take-off, departure, approach and landing times for UAVs that refuel. Section 8.3 and 8.4 then respectively discuss how the number of refuels are estimated for a particular scenario and how the flight schedules are consequently drawn up. Section 8.5 finally shows the familiar real-world examples with central deployment and endurance considerations.

8.1 Central Deployment Concept

A problem that has been mentioned throughout this project is the UAV endurance limitations. The algorithms that have already been discussed can be used when accounting for this.

The spanning tree algorithm, that is used to generate the coverage paths in the environment sub-regions, forms closed-loop paths. This means that a UAV in a given sub-region can be expected to end its path with the same heading and at the same coordinates as where it started.

These closed-loop paths work well with a central ground station concept. This means that the UAVs take off and land from a central location within the environment. With a central location for take-off, a single UAV can be launched easily to multiple initial positions in the environment at different times. A single UAV can therefore be used to search multiple sub-regions, with refuelling events in between.

The full path of the UAV, from the start of take-off to the end of landing, should take place within the endurance limitations of the UAV. If each single sub-region can be searched within the endurance limit of the UAV, it therefore

follows that the entire environment can be searched within the endurance limitations of the craft used. This presumes that all the UAVs in the environment have the same endurance limitations and that they always start their search fully fuelled or charged.

Refuelling in this chapter refers to any event, or sequence of events, necessary to get a UAV flight ready. This could be changing or recharging of batteries. It could also be a physical refuel for combustion engine powered craft.

With refuelling, a single UAV initial position will still be present within each sub-region after execution of the divide areas algorithm. However, multiple initial positions may represent a single physical UAV. For this chapter, the number of UAV initial positions will be referred to as the equivalent number of UAVs. These are not representative of the number of actual UAVs used to cover the environment.

To make use of this central ground station concept, the equivalent number of UAVs need to have initial positions assigned to them. These initial positions need to be spread out enough to ease the division of areas, while also being close enough to the ground station to enable moving to and from UAV initial positions in a reasonable amount of time.

In this project, the UAVs are clustered with at least one large cell in between them. It is possible to space them one cell apart, but then the divide areas algorithm would be more likely to take longer to execute, or fail to find a solution. The UAVs are also placed in the outer small cells within these large cells and shifted in a clockwise manner. The outer small cells are used so as not to place all the UAVs directly adjacent to an obstacle. This also allows them to be spaced slightly further apart.

Figure 8.1 shows example configurations. The first configuration on the left is used for between one and four equivalent UAVs. The one on the right is used for between five and eight UAVs. The internal area of this is considered to be the ground station and is viewed as an obstacle to the divide areas algorithm. It is assumed that the survivor would not be present in this region,

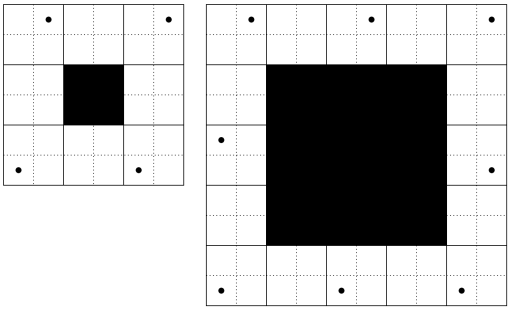


Figure 8.1: Diagram showing a central ground station with UAV configuration examples.

or that it is searched manually, if necessary.

For each increase in configuration size after this, the maximum number of UAV positions increases by four. Larger configurations should be used with care, particularly in smaller environments. It is not wise to place a ground station in an environment that occupies a large percentage of that environment.

Note that for a number of equivalent UAVs that is not a multiple of four, the UAVs' positions are simply filled in clockwise from the top left corner. It is possible to create countless alternative configurations for different numbers of UAVs, but the configurations in this project are designed to be as simple as possible. A detailed study into the different possibilities is not explored.

When placing a ground station and UAV initial positions in the environment, it should be placed logically. The search team should choose a wide open region where take-off and landing would be unobstructed. The take-off and landing are assumed to respectively end and start in the centre of this ground station. In other words, the craft reaches altitude at the centre of the ground station after take-off. Similarly, it starts its descent to the ground for landing from the centre of the ground station.

Flight schedules need to be calculated to make this ground station concept feasible. To start with, the problem of flight times can be addressed. Figure 8.2 shows an environment with eight equivalent UAVs. Each is treated as though it is its own craft and the DARP and STC algorithms are used to generate coverage paths.

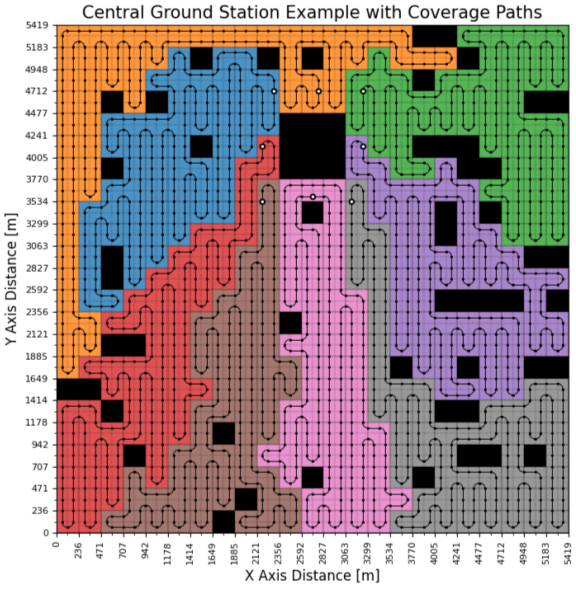


Figure 8.2: Example environment with coverage paths from a central ground station.

Due to the constant speed application, the path distances can be used to easily calculate the time taken to execute each path. The eight equivalent

UAVs, however, actually represent two physical UAVs that each refuel three times.

Figure 8.3 shows the sub-regions associated with each UAV in the same colour. To differentiate them, the shading of each sub-region is slightly different. This is to visualise which paths belong to the same aircraft. Note how there are now four UAV initial positions associated with each colour sub-region.

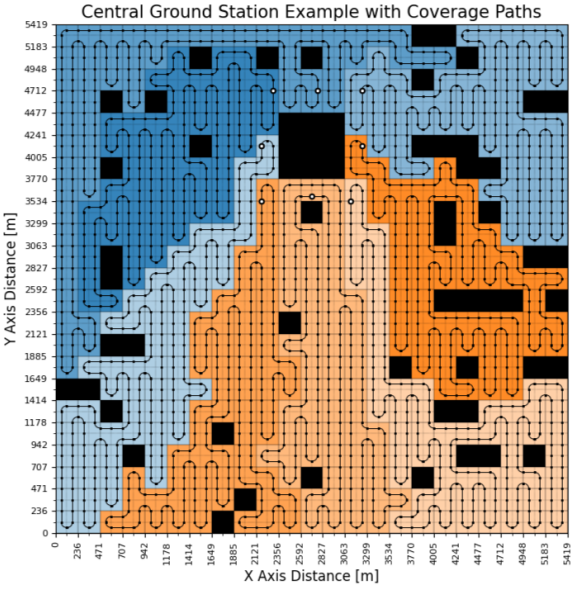


Figure 8.3: Example environment with coverage paths for two UAVs that refuel.

Take-off and landing times can be calculated to and from the centre of the ground stations, as mentioned. Departure and approach times are also calculated. These are related to the manoeuvre associated with moving to and from the initial positions of the equivalent UAVs. With these times as well as the flight times known, a flight schedule can be created. The assumption is that the UAVs take off one after the other and land in the same order. One UAV has to complete its take-off and departure before the next can begin its take-off. Similarly, one UAV has to complete its approach and landing before another can begin its approach. Similar length paths makes this a feasible schedule, without introducing too many wait times.

An airborne wait cycle would be a circular manoeuvre that the UAV executes while waiting for its turn to land. It may also wait on the ground before taking off, but this does not consume energy needed for flight.

The flight schedule for this example is shown in Figure 8.4. The legend describes what each colour represents. Each coloured region is representative of an action that consumes energy, except for the refuelling time.

The two UAVs are indexed as robot 0 and robot 1. Robot 0 takes off first, followed by robot 1. The small white regions on the schedule represent

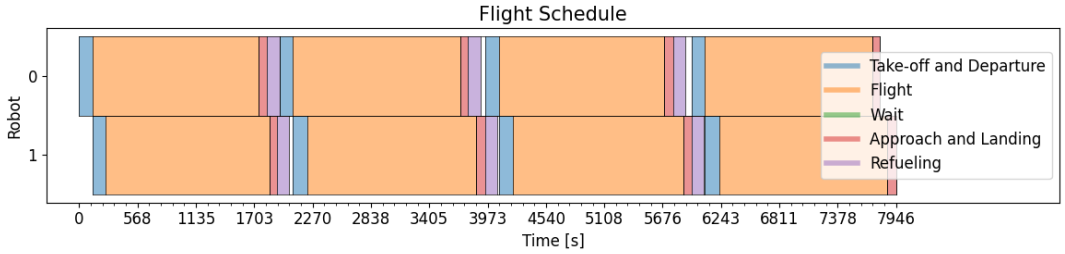


Figure 8.4: Flight schedule for the example environment.

time spent waiting on the ground prior to take-off. A green region would represent an airborne wait cycle. This schedule does not incur any of these wait cycles. To link the flight schedule back to the coverage paths in Figure 8.3, the blue sub-regions in this figure have been assigned to robot 0 and the orange sub-regions have been assigned to robot 1.

Note that this example utilises the Wingtra I UAV, which has rather short flight times. It is limited to about 42 minutes of flight time.

This is still an offline implementation. This means that the number of refuels must be calculated before generating the paths and schedule. After the algorithm is run it is then possible to check that the number of refuels is sufficient for the endurance limitations.

It is possible to therefore iteratively run the full algorithm until a feasible solution is found. However, in this project an accurate estimation of refuels is favoured over such an approach. This saves execution time when generating the algorithm.

Providing a ground station and endurance considerations makes it easy to see how these algorithms can be used for a real world SAR problem. Central deployment is an intuitive approach for a SAR operation using UAVs. Planning around endurance limitations is also crucial to ensuring the mission's success.

8.2 Time Calculations for UAV Manoeuvres

Departure and approach refer to manoeuvres between the UAV initial position and the centre of the ground station. Take-off would be the manoeuvre to reach the search altitude at the centre of the ground station and landing would be the descent to the ground from the centre of the ground station.

The times associated with all four of these manoeuvres need to be calculated for each equivalent UAV. This information is critical to later developing a feasible flight schedule.

The style of configurations for the UAV initial positions was shown in Figure 8.1. The smallest ground station size is the size of one large cell. The take-off and landing zone would be within this cell.

Figure 8.3 is labelled with an x-axis and a y-axis. Using this axis system, it is assumed for this project that the take-off occurs along the positive y-axis.

If north is at the top of the map, as is the general convention, take-off occurs in a northerly direction.

Once the UAV reaches altitude after take-off, it is assumed to have the take-off heading, at the centre of the ground station. The style of UAV would determine how this is achieved. It may take off vertically, from a runway or perhaps be launched by a mechanism or a person. Assuming it reaches altitude at the centre of the ground station, it follows that the take-off must be completed within one half of a large cell size, or simply the length of a small cell. It may be that this angle is too steep for a particular UAV. In this case the take-off region needs to be larger or some of the altitude should be gained during the departure.

For the purpose of this project, reaching altitude at the centre is assumed to be possible. The time to achieve this is a factor of only the altitude that needs to be gained and the climb rate of the UAV. The calculation is expressed by

$$T_T = \frac{H_f - H_T}{V_c} \tag{8.1}$$

where T_T is the take-off duration, H_T is the ground height from which the take-off occurs, H_f is the search altitude of the UAV, and V_c is the climb rate of the UAV. The landing time is calculated in a similar fashion. This is the time taken to descend to the ground from the centre of the ground station at the search altitude and is expressed by

$$T_L = \frac{H_f - H_T}{V_s} \tag{8.2}$$

where T_L is the landing duration, H_T is the ground height for take-off (which coincides with the landing ground height), and V_s is the UAV sink rate

Once again, it is assumed that landing is possible within a small cell length. If this is not possible, then the same rules would apply. Either the landing region needs to be larger or the descent should start during the approach.

The benefit of reaching altitude at the cell centre is that the time taken to reach the UAV initial positions from this point would be similar for all the equivalent UAVs. The time to reach the initial positions from the centre of the ground station is referred to as the departure time (T_D).

Calculating the departure and approach paths is done using an algorithm developed by Milne at Stellenbosch University [79]. The algorithm was originally developed in Matlab, but was reworked into Python for this application. The algorithm takes two waypoints as inputs. The UAV is assumed to fly at a given speed. With the initial and final positions, initial and final headings, and constant speed known, a path is generated to get from the one point to the other.

First, the departure path is generated to take the UAV from the take-off position and heading to the initial position and heading of the search path.

The time it takes to execute this path is called the departure time (T_D). After departure, the planned coverage path for that sub-region can be carried out. Since this is a closed-loop path, the final position and heading of the UAV are the same as its initial position and heading.

Following the coverage path is the approach path, which is calculated from the final position of the coverage path to the centre of the ground station using the same algorithm as with the departure. The time to execute the approach path is known as the approach time (T_A). This path starts with the same position and heading as what the departure path ended with.

Both the departure and approach manoeuvres are assumed to be carried out at the search speed of the UAV. The constant speed that is used for these manoeuvres is therefore the same as what is used for the coverage paths. The same dynamic constraints would apply during rotations.

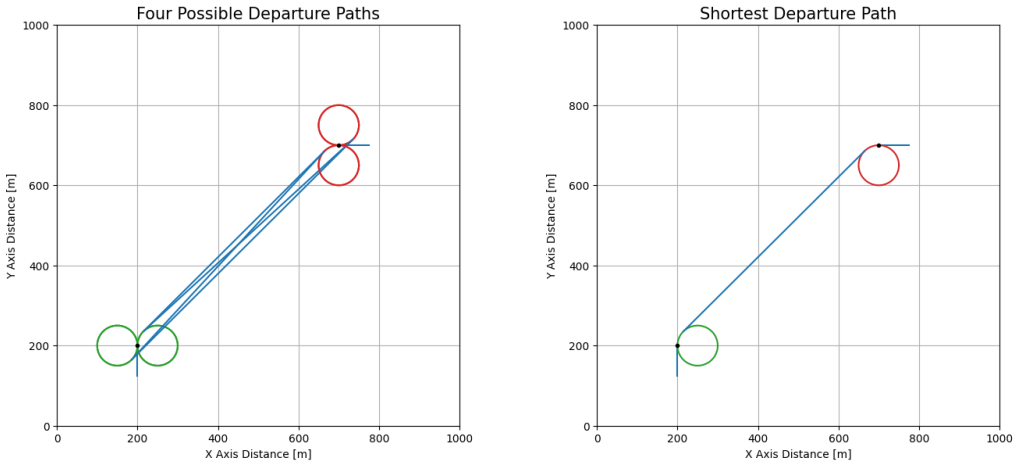
To illustrate how this algorithm functions, an example with two points is shown below. The centre of the ground station is placed at a distance of 200 m on the x and y axes. The runway, if applicable, is assumed to be aligned with the y-axis. A UAV initial/final position is then represented by the point at a position of 700 m on both axes. The algorithm connects the two waypoints using a straight line tangential to two arcs representing the minimum turning radius of the UAV. This ensures that the UAV can reach the desired heading at the end of its path.

After take-off, the starting heading is chosen to be in the direction of the positive y-axis. For this example, the desired final heading of the UAV, at the initial position for the coverage path, is assumed to be in the direction of the positive x-axis. Assuming in this case that the speed limits the turning radius of the UAV to 50 m, four possible paths are generated. These are depicted in Figure 8.5a. The shortest of these is then selected, which is depicted again in Figure 8.5b.

Both waypoints are shown in the figures. The green circle represents the starting arc and the red circle represents the ending arc on the path. The small straight lines extending from the arcs are the heading vectors at both waypoints.

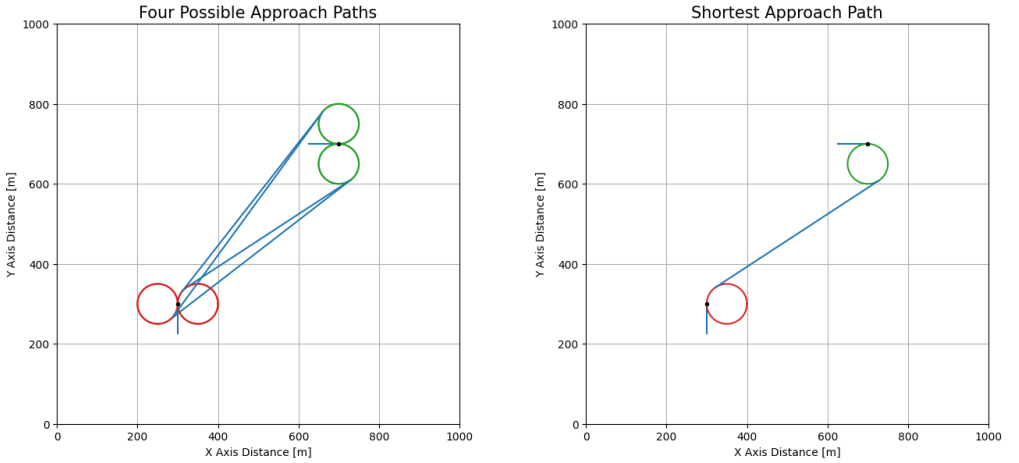
The same method is used to generate the approach paths. Figure 8.6a shows the four possible approach paths. Figure 8.6b then shows the shortest path of the four that gets selected.

The positions of the start and end waypoints are opposite to that during departure. Due to the closed-loop coverage path, the approach starts where the departure ended, with the same heading along the positive x-axis. At the end of the approach, the UAV is at the start of its descent from the search altitude to the ground. The heading at the end waypoint, at the centre of the ground station, is therefore assumed to be in the direction of the negative y-axis. This would be its landing heading. Both heading vectors are shown on the figures.



(a) Four possible departure paths. (b) Shortest departure path.

Figure 8.5: Diagram showing the four possible departure paths for the UAV and the selected shortest path.



(a) Possible approach paths for a UAV (b) Shortest approach path for a UAV.

Figure 8.6: Diagrams showing the four possible aaproach paths for the UAV and the selected shortest path.

Now that the path lengths for departure and approach are known, they can be used to calculate the times associated with them. The calculations for departure time and approach times are as follows,

$$T_D = \frac{P_D}{V_f} \tag{8.3}$$

$$T_A = \frac{P_A}{V_f} \tag{8.4}$$

where P_D and P_A are the path lengths for the departure and approach manoeuvres respectively. T_D and T_A are the corresponding durations for these manoeuvres, and V_f is the constant flying speed of the UAVs.

Determining the heading of the UAV at the start and end of its coverage path is done using the half shifts that are used when formulating the coverage paths. The coverage paths are executed clockwise, but the half shifts are counter clockwise. The heading is therefore always in the opposite direction to the shift relative to the small cell centre.

Figure 8.7 shows how this works. The top square represents a large cell. A UAV initial position (the black dot) is in the top right small cell of the large cell. The bottom images depict the four possible half shifts of this initial position within the small cell. In each case it is clear that the heading, represented by the arrows, is in the opposite direction to the shift.

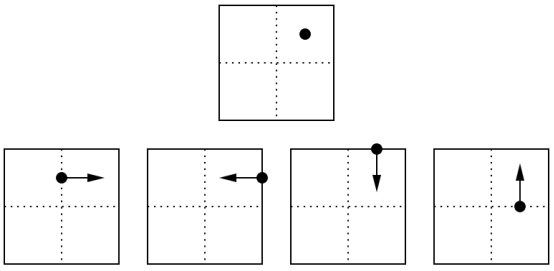


Figure 8.7: Diagram showing relationship between heading and half shifts.

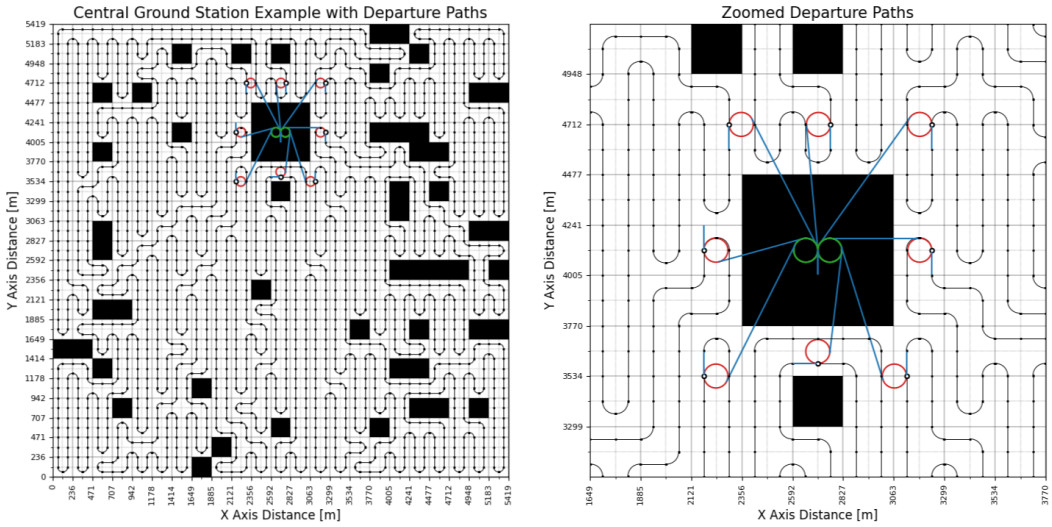
The departure and approach paths developed by the algorithm mentioned can be overlaid onto the example of Figure 8.3. Figure 8.8a shows the departure paths for the eight equivalent UAVs of this example. For clarity, the sub-region colours are removed in the figure. A zoomed in view is also shown in Figure 8.8b to more clearly see the headings when they start their coverage paths.

Note that these departure paths would not all be executed at once, but one after the other. In practice, the starting position may not be in the centre of the ground station. It would depend on a number of factors and may be unique to a scenario.

There are certain edge cases wherein collisions could occur between UAVs. There is assumed to be an online, short-term collision avoidance system onboard each UAV. This system should be designed to avoid dynamic obstacles in the environment, including other UAVs, while adhering as closely as possible to the original planned path [16].

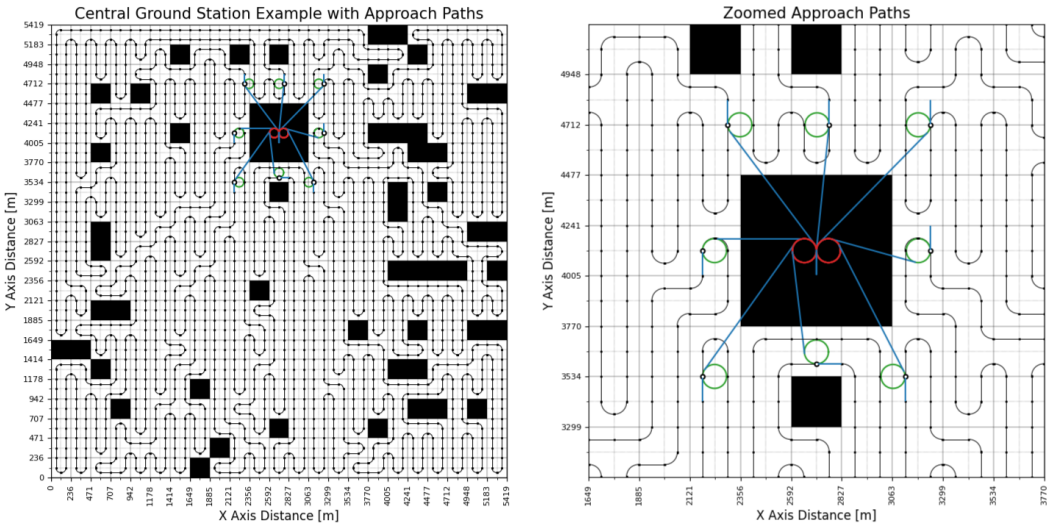
The approach paths are also shown in Figures 8.9a and 8.9b. Note how the headings at the start of these paths are the same as those at the end of the

departure paths. Once again, green represents the arc where the path begins and red represents where it ends.



(a) Departure Paths. (b) Zoomed Departure Paths.

Figure 8.8: Departure paths for UAVs in the example environment.



(a) Approach Paths. (b) Zoomed Approach Paths.

Figure 8.9: Approach paths for UAVs in the example environment.

8.3 Endurance Estimation

Flight schedules are generated after the coverage paths have already been determined. Therefore, before exact schedules and times are known, the number of sub-regions should already be known. The number of sub-regions are equal to the equivalent number of UAVs, and these are a function of the number of refuels. It follows then that the number of refuels for the UAVs should be known prior to formulating the coverage paths.

This section discusses the technique used to determine the number of refuels that will be necessary in a specific scenario. Figure 8.10 depicts the logic used in the form of a flow diagram. The first step for calculating refuels is determining the number of cells that can feasibly be assigned to each UAV.

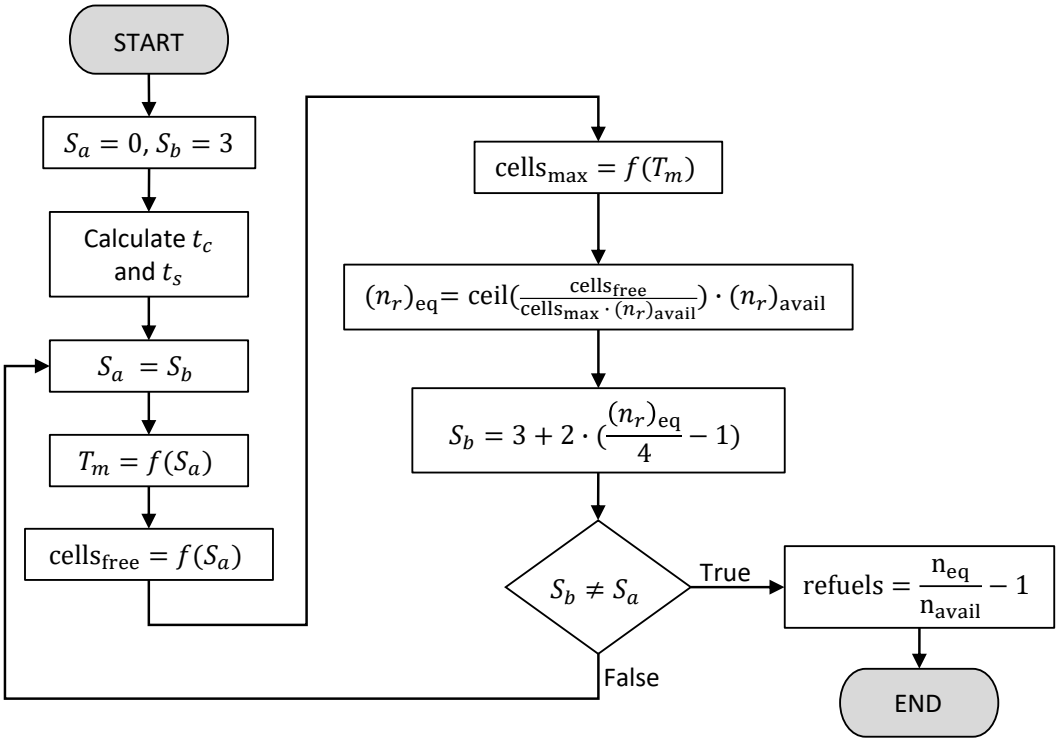


Figure 8.10: Flow diagram showing the logical progression for calculating the number of refuels.

Equation 7.5 can be used to calculate the maximum number of cells that can be assigned to a single UAV. The variable T_m in this equation represents any manoeuvres outside of the coverage flight time that consume energy. This includes take-off, departure, wait, approach, and landing times. A safety factor is applied to all of these to account for increased energy consumption due to the changes in altitude and rotations. The calculation is reiterated with the

details of the T_m variable expanded,

$$\text{cells}_{\max} = \frac{T_p - k_t(T_T + T_D + T_W + T_A + T_L)}{\max(k_t t_c,\ t_s) \cdot 4} \tag{8.5}$$

where T_T, T_D, T_W, T_A and T_L are the take-off, departure, wait, approach and landing times, and k_t is the safety factor. T_p is the predicted flight time of the UAV, t_c is the time it takes to execute a 90 degree turn, and t_s is the time it takes to execute a straight line manoeuvre in the confines of a small cell. The denominator as a whole is the maximum time it can take to cover a large cell, which consists of four small cells.

Most of these times need to be estimated in order to get the number of refuels. Once the number of refuels is known, the actual times can be calculated. The energy consumption can then be checked to ensure that it is within the limits of the vehicle. Having to rerun the algorithm is undesirable though, so this implementation seeks to make fairly accurate estimates of these times. The take-off and landing times are known and do not need to be estimated. They are a function of the search altitude and the climb and sink rates of the UAV. They can be calculated using Equations 8.1 and 8.2.

The departure and approach times can be approximated as two arcs and a straight line. To be conservative, the arcs are assumed to be full circles. The straight line distance is also conservatively assumed as the distance from the centre of the ground station to the edge of a corner starting cell. This approximation is depicted in Figure 8.11. Note that the arcs are a function of the minimum turning radius of the UAV (r_{min}) and the straight line distance is a function of the small cell length (l).

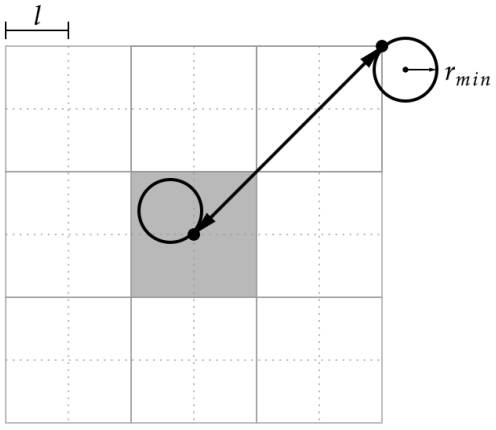


Figure 8.11: Diaram showing the approximation used for a departure or approach path length.

The total distance of the circles and the straight line is divided by the cruising speed of the UAV at the search altitude. This gives an estimated

departure or approach time given by

$$T_D \approx T_A \approx \frac{4\pi \cdot r_{\min} + \sqrt{2}\, S \cdot l}{V_f} \tag{8.6}$$

where T_D and T_A are the approximate times to complete the departure and approach manoeuvres respectively, $r_{\min}$ is the minimum turning radius of the UAV, l is the dimension of a small cell, and V_f is the constant flying speed of the UAV. The S variable represents the number of large cells that represent the square occupied by the UAV initial positions. Note that the time estimate is a factor of the ground station size. For Figure 8.11 this value would be three since it is a three-by-three large cell grid.

A wait cycle is incurred only when one UAV wants to start its approach, but another UAV is still busy with their approach or landing. Wait cycles are always one full circle at the minimum turning radius. This ensures that the starting point and heading of the UAV, as it starts its approach, remain constant. This circle coincides with the starting arc of the approach manoeuvre. These manoeuvres do not often occur and so one wait cycle is assumed to be a reasonable estimate. The estimated wait time calculation is given by

$$T_W \approx \frac{2\pi \cdot r_{\min}}{V_f} \tag{8.7}$$

where T_W is the approximate time to complete an airborne wait cycle, $r_{\min}$ is the minimum UAV turning radius, and V_f is the constant flying speed of the UAV.

The departure and approach times are a function of the ground station size. Similarly, the number of free cells in the environment also change depending on this size. The free cells represent the total number of cells in the environment with the number of obstacle cells subtracted. Since the ground station is viewed as an obstacle it would then change the number of free cells when it is added. The size of the ground station can be represented by a value two less than the size variable mentioned previously. The free cells can be represented as a factor of this since all other obstacles in the environment are constant. The free cells can then divided by the the maximum number of cells that can be assigned to one UAV. This gives the equivalent number of UAVs needed to search this environment.

Once again, the value for the equivalent number of UAVs is adjusted to be an integer multiple of the number of available UAVs. The overall expression is given by

$$\begin{aligned} (n_r)_{\text{eq}} &= \frac{N_t - N_o - (S-2)^2}{\text{cells}_{\max}} \\ (n_r)_{\text{eq}} &= \text{ceil}\left(\frac{(n_r)_{\text{eq}}}{(n_r)_{\text{avail}}}\right) \cdot (n_r)_{\text{avail}} \end{aligned} \tag{8.8}$$

where N_t is the total number of cells in the environment, N_o is the number of cells in the environment that are classified as obstacles, and $(S-2)^2$ is the number of cells that make up the ground station. $(n_r)_{\text{eq}}$ is the equivalent number of UAVs which is the same as the number of sub-regions in the environment; $(n_r)_{\text{avail}}$ is the number of available UAVs; and $\text{cells}_{\text{max}}$ is the maximum number of cells that a UAV can search before needing to refuel. The flow diagram of Figure 8.10 shows the equation with the notation prior to expanding the definition of the free cells. It also merges the two-step equation into one.

This is where the algorithm becomes iterative, since the equivalent number of UAVs can change the size of the ground station. Therefore, the way this algorithm is executed is by assuming a ground station size. Initially it is assumed to have a size of one cell, since this is the smallest configuration. The size variable S in turn becomes three cells initially, as shown on the flow diagram.

The calculations are executed up until the equivalent number of UAVs can be found. If this number of UAVs demands a bigger ground station, the calculation is restarted at this new ground station size. Figure 8.10 clearly shows the loop formed from this logic. The algorithm converges, and so the size will eventually no longer be adjusted. The number of refuels can then be calculated.

Refuels are calculated using Equation 7.7 which is reiterated in the flow diagram. With the refuels known, the algorithm terminates. The actual energy consumed can then be calculated when the algorithm is run with this number of refuels. It should then be confirmed that the energy consumption is within limits.

Note that no special considerations in terms of energy consumption are made for undesirable weather conditions. However, it is possible to add an overall safety factor to the predicted flight time (T_p) of the UAV to accommodate this in a real world scenario.

8.4 Flight Schedule and Survivor Detection

This section works through the process of creating a full flight schedule for an environment. The first example shown in this section is the one that has been carried throughout this chapter. A second example is also shown to give insight into the variations that may occur in a flight schedule. For the examples in this section, the ground was assumed to be level. For a real-world scenario, this would likely not be the case. The flight schedule generated in this section is also a preliminary one. For real-world application, some adjustments may be necessary.

Once the number of refuels for a specific environment are known, the initial positions of the equivalent UAVs are placed. The divide areas algorithm

then assigns sub-regions to each equivalent UAV. The coverage path lengths for each region can then be determined using the sub-region coverage technique.

With the paths known, the starting headings are known at the UAV initial positions. The take-off, departure, approach, and landing times are then calculated according to the methods described in Section 8.2. Furthermore, the actual flight times can be calculated. This would be the sum of all the manoeuvres executed during the coverage path, each multiplied by the time to execute them. In this case it is the number of straight line manoeuvres (n_s) and 90 degree circular manoeuvres (n_c), multiplied by the times to execute them. The relevant calculations were shown in Equations 7.1 through 7.3.

The total time to execute a sub-region search is calculated according to

$$T_{\text{tot}} = T_T + T_D + T_W + T_A + T_L + n_s t_s + n_c t_c \tag{8.9}$$

which excludes the time used to refuel and the time spent idle on the ground in between searches, but includes the time to execute the coverage path. Using the same variables, but with the safety factors applied as necessary to represent energy consumption during flight, an expression for total energy consumption can be created. This is given by

$$(T_{\text{tot}})_e = k_t \cdot (T_T + T_D + T_W + T_A + T_L) + n_s t_s + k_t \cdot n_c t_c \tag{8.10}$$

where T_{tot} is the total time to execute a sub-region search and $(T_{\text{tot}})_e$ is an approximation of the energy consumption during a sub-region search. T_T, T_D, T_W, T_A and T_L are the respective times to complete take-off, departure, any airborne wait cycles, approach and landing. The number of straight-line manoeuvres in the coverage path is n_s, and the time to execute a straight line manoeuvre is t_s. The number of 90 degree turns is n_c and the time to execute one turn is t_c. k_t is a factor applied to account for an increase in energy consumption during certain manoeuvres, that may decrease the predicted flight time.

Note that each equivalent UAV now has an associated take-off, departure, flight, approach, and landing time. With these known, the schedule can be designed. A UAV is selected to be the first one to take off. It is only assumed to be safe for the next UAV to take off once the first UAV has completed its take-off and departure manoeuvres.

For this reason, take-off and departure are grouped into one category in Figure 8.12. The blue regions show this time. Once a UAV has reached its initial position it starts on its coverage path. The time to execute the coverage paths are represented by the orange regions on this flight schedule. Once the first UAV completes a coverage path, it immediately starts with its approach and landing manoeuvres. Refer to robot 0 on the figure. The landing and approach is grouped into the same category and is represented by the red region.

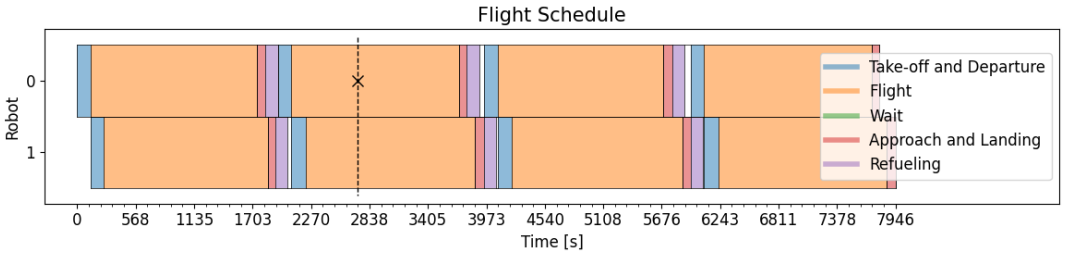


Figure 8.12: Flight schedule for the example environment with survivor detection indicated.

Note that the next UAV cannot start its approach and landing until the previous one has completed theirs. If a UAV finishes its coverage path before the previous UAV has finished landing, it is not permitted to land yet. It would have to execute an airborne wait cycle. This specific situation does not occur on this example flight schedule. Note the absence of green regions. The advantage of having similar coverage path lengths, as well as similar approach and departure times, is that wait times seldom occur.

The purple region on the flight schedule indicates the time taken to refuel or recharge the UAV once it has landed. For this example it was chosen as two minutes. Particularly since this scenario would only require a battery change, this is expected to be sufficient. The actual time needed to complete a refuel would have to be tested for a specific UAV in a real-world scenario. It would also be advisable to allocate more time than necessary to this activity, to ensure the UAVs can adhere to their planned schedules.

The flight schedule also has white regions. These simply represent time where the UAV is idle on the ground, waiting for its turn to take off. One UAV needs to complete its take-off and departure before another can begin. The last UAV also needs to have finished landing before the first UAV in the schedule can take off again.

Figure 8.13 shows the execution of the planned flight schedule by the two UAVs up to the point where the survivor is found. The target is placed at a 200 m distance on the x-axis and a roughly 2.5 km distance on the y-axis. A snapshot is taken at the time the target it found. This time is also indicated on the flight schedule with a dotted line. The "**X**" indicates which UAV detects the target at this time. In this case, the survivor is found in a blue sub-region, representing robot 0 on the schedule. The detection occurs after the first refuel. Both UAVs have completed their first sub-region search and the survivor detection occurs during the second sub-region search.

Completing the full flight schedule would take about 2 hours and 13 minutes. The target, in this case, is located around 45 minutes into the search. The order in which the sub-regions are searched determines how soon a target is found. This order is standardised for this algorithm. It is possible to change which regions get searched first depending on information of the target's most

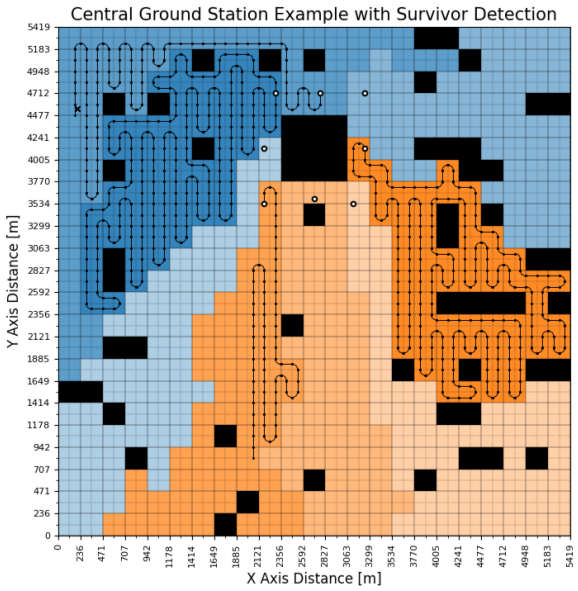


Figure 8.13: Snapshot of example environment at the point of target detection.

likely location. This is not explored further in the scope of this project.

A second example environment is shown to illustrate a scenario where a wait cycle is introduced. Figure 8.14 shows the resulting flight schedule for this example. The target is found by the second UAV, after having refuelled

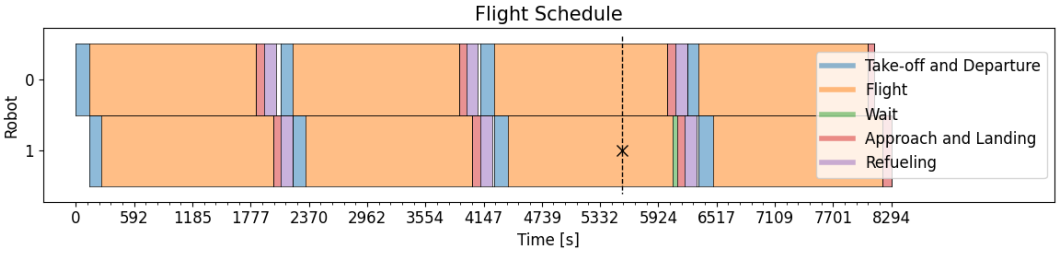


Figure 8.14: Flight schedule for the second example environment with survivor detection indicated.

twice. Note that for the full plan, a wait cycle is induced for the second last landing event. This is represented by the green section on the schedule. This time represents one full circle of flight at the minimum turning radius. This ensures that the first UAV finishes its approach and landing before the second UAV begins its approach. To ensure the start position and heading during approach are constant, the wait cycles are always in multiples of the time to execute one 360 degree turn during flight.

The map of the second example environment, with the coverage paths generated, is shown in Figure 8.15. Figure 8.16 shows the execution of the planned flight schedules for the two UAVs up to the point where the survivor is found. The survivor was placed at 20 m in the x-axis and about 1 km on

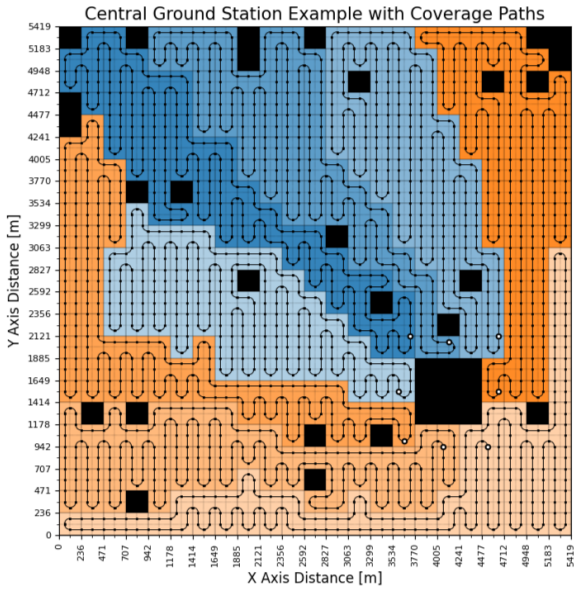


Figure 8.15: Second example environment with coverage paths depicted.

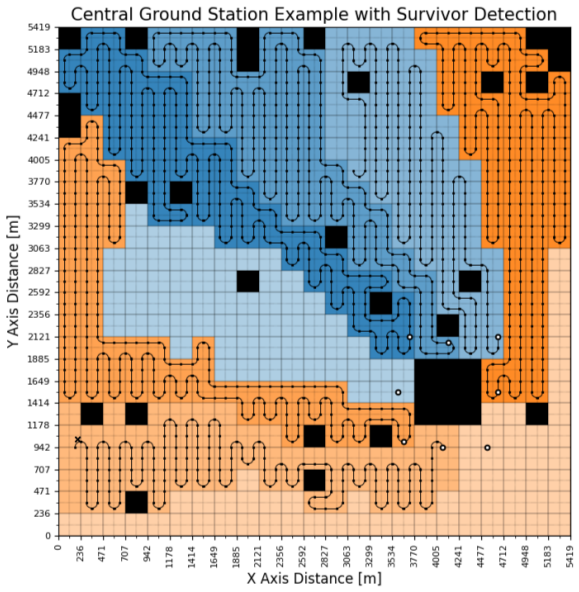


Figure 8.16: Snapshot of second example environment at point of target detection.

the y-axis. Executing this full flight schedule would take about 2 hours and 18 minutes. Survivor detection occurs roughly 1 hour and 33 minutes into the search. Figure 8.16 shows a snapshot of the paths at this time.

Note how both UAVs completely search two sub-regions before the target is found. It is located well into the third orange sub-region. The orange regions represents robot 1 in the flight schedule.

The algorithm for the refuelling problem also outputs a tabular format of

the data. Figure 8.17 shows the tabulated data for this example. The data is divided amongst two tables for the refuelling output. In each table, the index of the physical UAV and the number of refuels it has had are indicated.

The first table shows all the times used for the flight schedule. The sum of these times gives the total time from take-off through to landing. The distance and rotations columns are values associated with the coverage path in particular. This table is shown in Figure 8.17a.

The second table indicates the total time it takes each UAV to complete its path from the moment it leaves the ground to when it lands again. The associated energy value, where the safety factor is included, is also shown. The time excess value is then an indicator of whether the flight plan is within the energy constraints of the UAV. This is the final check to see if the generated plan is feasible. In this case it is, as can be seen in Figure 8.17b. The time excess values are around seven minutes, which is also a good buffer to ensure the UAVs can actually complete their planned routes.

Robot	Refuel	Take-off [min]	Departure [min]	Flight [min]	Wait [min]	Approach [min]	Landing [min]	Distance [km]	Rotations
0	0	1.4	0.9	28.3	0.0	0.6	0.8	27.2	86.0
0	1	1.4	0.6	28.3	0.0	0.6	0.6	27.2	66.0
0	2	1.4	0.9	29.2	0.0	0.6	0.8	28.1	48.0
0	3	1.4	0.6	28.6	0.0	0.6	0.6	27.4	74.0
1	0	1.4	0.6	29.3	0.0	0.6	0.6	28.1	46.0
1	1	1.4	0.9	28.1	0.0	0.6	0.9	27.0	72.0
1	2	1.4	1.0	27.8	0.7	0.6	0.8	26.7	84.0
1	3	1.4	1.0	28.8	0.0	0.6	0.9	27.6	66.0

(a) Table showing the time values associated with each phase in in the flight plan of the UAVs.

Robot	Refuel	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]
0	0	31.9	35.5	42.0	6.5
0	1	31.4	34.3	42.0	7.7
0	2	33.0	35.5	42.0	6.5
0	3	31.8	34.9	42.0	7.1
1	0	32.5	34.8	42.0	7.2
1	1	31.9	35.1	42.0	6.9
1	2	32.3	36.1	42.0	5.9
1	3	32.6	35.7	42.0	6.3

(b) Table showing the values necessary to determine the feasibility of the flight plan for the UAVs.

Figure 8.17: Tables showing algorithm outputs for example environment

8.5 Illustrative Examples with Different Environments

This section presents several of the example environments that have been used throughout this project. Here they are shown in the context of the refuelling problem. In each case, a limited number of UAVs are provided so that refuelling would be required.

The figures showing the coverage paths and survivor detection are shown at the end of this section. The flight schedules and data table for each scenario are also shown for each example, as they are discussed. The actual approach and departure paths are not depicted on the figures of this section, but are considered implicitly present.

Spitskop

The first example is the Spitskop environment. This environment made use of a Strix 400 UAV in the original problem. Due to this UAV's long endurance capabilities, it can search quite large areas before needing to refuel. The examples given throughout this chapter so far have made use of the Wingtra I, which is more suitable for smaller areas.

For this example, the number of available UAVs were selected as two. Having only two UAVs available is very possible for a SAR scenario. For this case the ground station is placed at a 12 km distance on the x-axis and a 1 km distance on the y-axis.

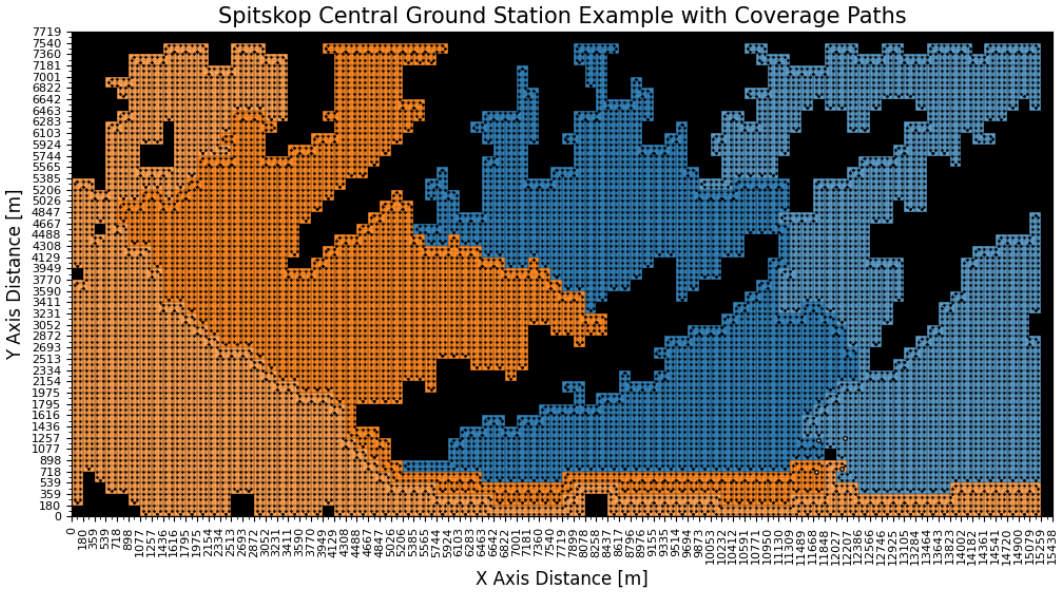
With this ground station location, the take-off height is roughly 350 m above sea level. The take-off and landing times can be calculated using this and the desired flying altitude of 660 m above sea level for the search. Table 8.20a shows the take-off time as just over a minute and the landing time as just under a minute long.

Using the iterative technique described in Section 8.3, it was found that one refuel should be sufficient to cover the demarcated search area with two UAVs. This brings the equivalent number of UAVs to four. With this many equivalent UAVs, the ground station is the size of one large cell. The four initial positions of the UAVs are placed around this ground station as shown in Figures 8.18a and 8.18b.

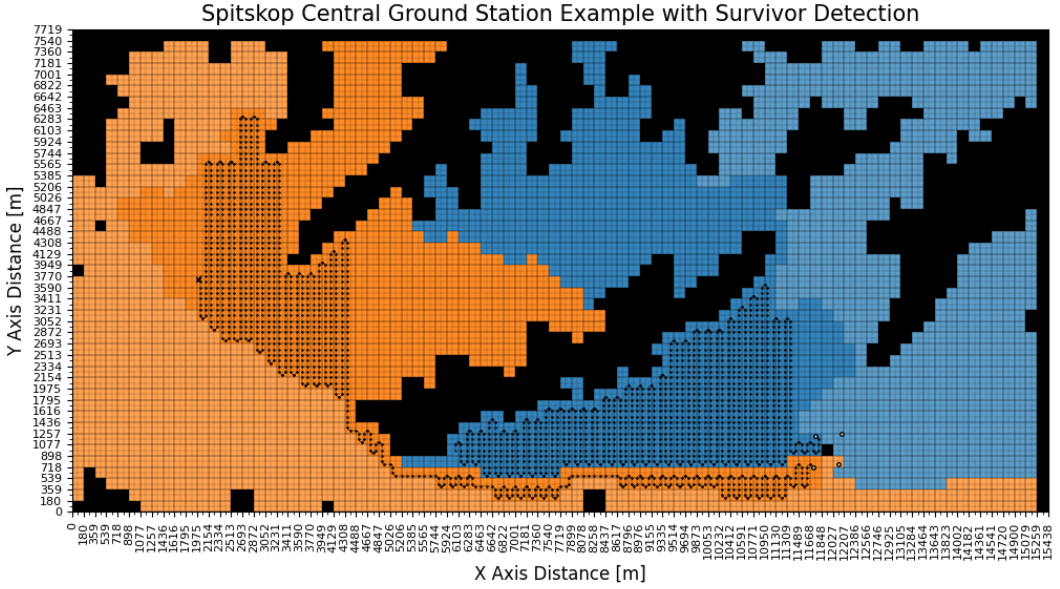
The initial positions are similar to the clustered scenario shown in Figure 7.24. They are once again shown as a black dot with a white dot at its centre. With these initial positions known, the sub-regions were generated with the divide areas algorithm. Two of these sub-regions were then assigned to each UAV. The regions of the same colour belong to the same UAV in Figures 8.18a and 8.18b. They are shaded slightly differently to still make them distinguishable. Within these sub-regions, the coverage paths are generated using the sub-region coverage technique. Figure 8.18a shows these paths.

The time to execute these respective paths at the 14 m/s constant speed of the UAVs is shown in the "Flight" column of the table in Figure 8.20a. The "Distance" and "Rotations" columns are also associated with the coverage path. They represent the physical length of the coverage path in kilometres and the number of 90 degree rotations within the path.

The coverage path generation results in the half shifts of the UAV initial positions, which make it easier to determine the headings of the UAVs at their



(a) Coverage paths for Spitskop example environment with a central ground station.



(b) Snapshot of the Spitskop example environment at the point of target detection

Figure 8.18: Spitskop example environment with two UAVs that refuel.

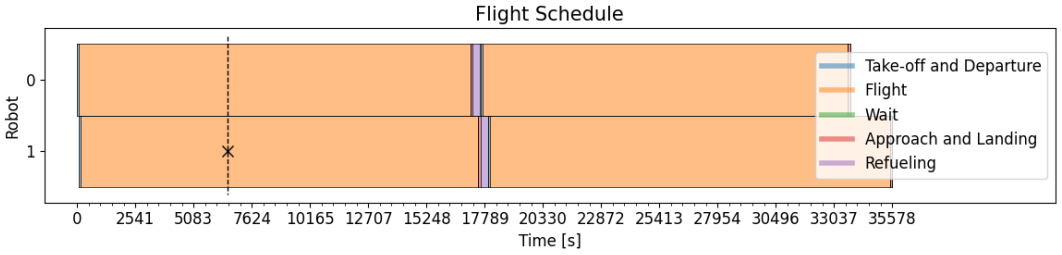
initial positions. With this known, the approach and departure times can be calculated. These are also detailed in the table.

The table entries for the tables in Figure 8.20 have four rows, representing each equivalent UAV. The first column of both tables shows the index of the actual UAV. The second column then references how many refuels have occurred for the physical UAV at this point.

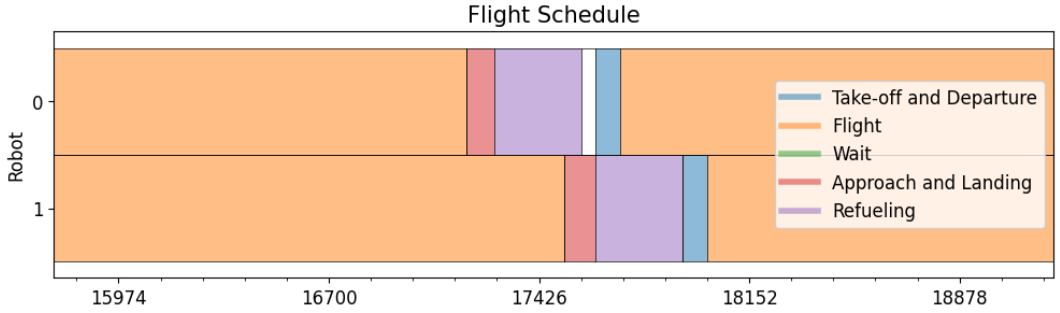
With the take-off, departure, flight, approach and landing times known, a flight schedule can be drawn up. During the creation of this schedule, it may

be found that airborne wait times are necessary. In this case, no wait times are induced and so the "Wait" column in the table of Figure 8.20a has zero values.

The resulting flight schedule is shown in Figure 8.19a. Here the one refuelling event can clearly be seen for each UAV. A more detailed view of this refuelling event for both UAVs can be seen in Figure 8.19b. The purple regions represent the five minute refuelling time that is allocated for this UAV. In a real-world scenario, these schedules may need to be adjusted based on data collected regarding the UAV's performance, and the time taken to replace its battery, which represents a refuelling event. Slightly more time is given for refuelling the Strix 400 over the Wingtra I. It is a larger UAV with bigger batteries, so it is reasonable to assume that changing out the batteries would be more time consuming.



(a)



(b)

Figure 8.19: Flight schedule for the Spitskop example environment with two UAVs that refuel.

There are no green regions, meaning that there are no airborne wait cycles. The white region indicates a time where the first UAV waits on the ground. It can only take off once the second UAV has completed its landing.

The total time taken for each equivalent UAV to complete its cycle from the start of take-off to the end of landing is shown in the third column of the table in Figure 8.20b. In this case the time to complete the full flight plan is about 9 hours and 53 minutes.

Robot	Refuel	Take-off [min]	Departure [min]	Flight [min]	Wait [min]	Approach [min]	Landing [min]	Distance [km]	Rotations
0	0	0.9	0.4	284.9	0.0	0.4	1.3	239.3	418.0
0	1	0.9	0.5	265.4	0.0	0.4	1.3	222.9	352.0
1	0	0.9	0.6	289.0	0.0	0.5	1.3	242.8	446.0
1	1	0.9	0.5	291.1	0.0	0.5	1.3	244.5	448.0

(a)

Robot	Refuel	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]
0	0	287.9	299.4	540.0	240.6
0	1	268.6	278.5	540.0	261.5
1	0	292.4	304.7	540.0	235.3
1	1	294.3	306.6	540.0	233.4

(b)

Figure 8.20: Data tables generated for the Spitskop example environment with two UAVs that refuel.

The target is at a distance of 2 km on the x-axis and 3.7 km on the y-axis, as it was previously in Section 7.5. A snapshot of when the target is located, with central deployment and refuelling, is shown in Figure 8.18b. The time at which this occurs is roughly 1 hour and 49 minutes into the search. The detection occurs in the first orange region that is searched. If this were the second region searched, the time would be significantly longer.

Although refuelling allows the area to be searched with only two UAVs, having four UAVs searching in parallel would still be much preferred. In general, having more UAVs available increases the odds of finding the survivor(s) sooner for any scenario. The target detection event is also indicated on the flight schedule with a dotted line and an "**X**" that shows it was detected by the second UAV on the schedule, prior to refuelling.

To convert the total time for searching to a value representing energy consumption, a safety factor of 1.3 is applied to certain manoeuvres. The next column shows the energy value that is calculated. The energy value represents the energy required to completely search a single sub-region. The "Time Limit" column then represents the amount of energy available. Since the Strix 400 UAV is rated for 9 hours of flight at the lower end of its bracket, this value is 540 minutes.

The "Time Excess" column gives the difference between the available energy and the total energy used. Therefore, a positive excess value indicates a feasible flight plan. In this case, there is an excess of around 4 hours in each case, which is more than sufficient.

Champaigne Castle

The Champaigne Castle example was also originally searched with a Strix 400. With this UAV's endurance, only one UAV could cover the entire region without refuelling. Therefore it is not explored as a possible refuelling example.

Aberdeen

The Aberdeen example, on the other hand, shows an example of the opposite problem. This is a large region that was originally going to be searched using the Wingtra I. However, this UAV has an endurance of just 42 minutes.

The number of equivalent UAVs would be a multiple of the available UAVs. Assuming there are 2 UAVs available, the equivalent UAVs become 28. This results in a ground station size of 9 by 9 large cells. This can be used, but the size is impractical. It would also not be advisable to try and optimally divide a region into 28 sub-regions using the iterative divide areas algorithm. The Aberdeen scenario therefore shows why long endurance craft are so valuable when it comes to large search regions. Endurance should also be factored in when choosing the UAV to search an area. The Wingtra I is more suitable for smaller areas. Since the Strix 400 has a lower cruise speed than the Wingtra I, it can be used instead of the Wingtra I with the same discretisation. It is assumed that three UAVs are available to search this area. The ground station is positioned at 5.5 km on the x-axis and 2.5 km on the y-axis.

The take-off height at the ground station is about 790 m above sea level. The flying altitude is 1060 m above sea level. The difference between this is the height used in calculating the take-off and landing times.

The endurance estimations can be used to determine the number of refuels, which is just one. The number of equivalent UAVs then becomes six. These are arranged around the ground station as shown in Figure 8.21a. This figure also shows the resulting coverage paths. Figure 8.21b shows the target at a distance of 6.2 km on the x-axis and 4.7 km on the y-axis. This figure shows a snapshot at the moment of survivor detection.

With the paths known, all the relevant times can be calculated to generate the flight schedule. The relevant times are shown in the table of Figure 8.23a.

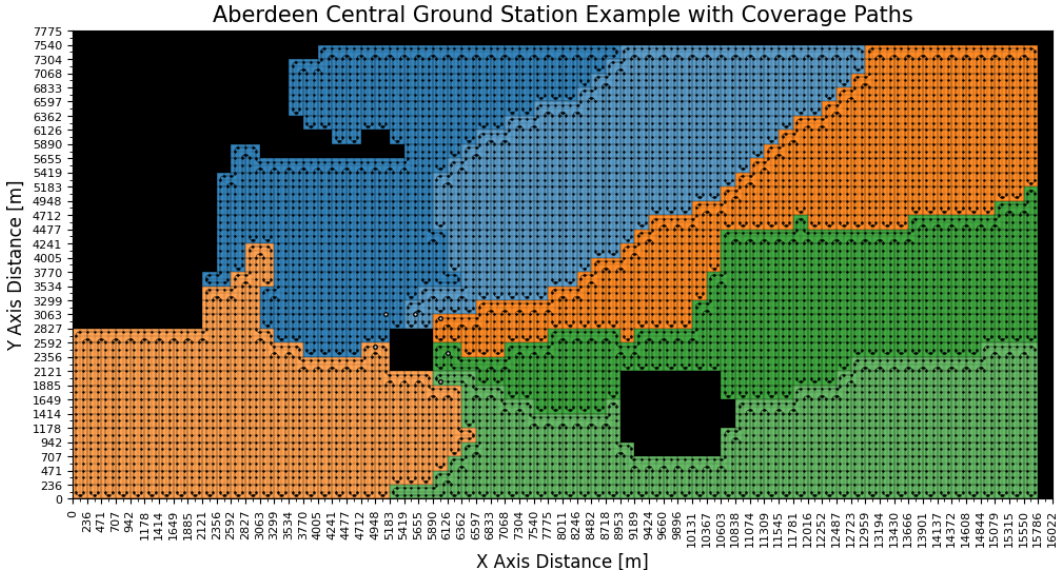
This scenario causes airborne wait cycles, unlike the Spitskop example. The full flight schedule is shown in Figure 8.22a. The refuelling event is shown in more detail in Figure 8.22b, where the green regions are visible.

The green regions represent a wait cycle. The UAV waits in the sky by flying in a circle. The wait is induced because a UAV is busy landing when another finishes its coverage path. In this case, robot 1 has to execute one wait cycle to allow robot 0 to finish its landing, before starting its approach. Similarly, robot 2 has to execute two wait cycles while waiting for robot 1.

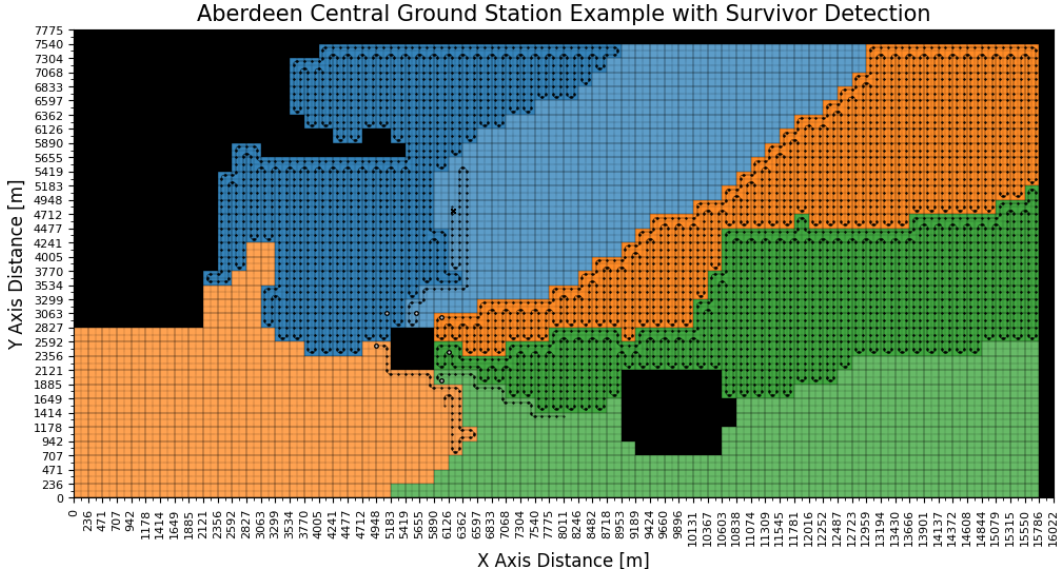
The table of Figure 8.23a shows these wait times. One wait cycle takes just over 30 seconds to complete. Two wait cycles take about a minute. They will always be multiples of the time taken to complete one cycle.

The table of Figure 8.23b shows the energy values. Once again, the excess values are positive, indicating a feasible flight plan. The tables have six entries this time, representing the six equivalent UAVs.

The total flight schedule for the Aberdeen environment would take about

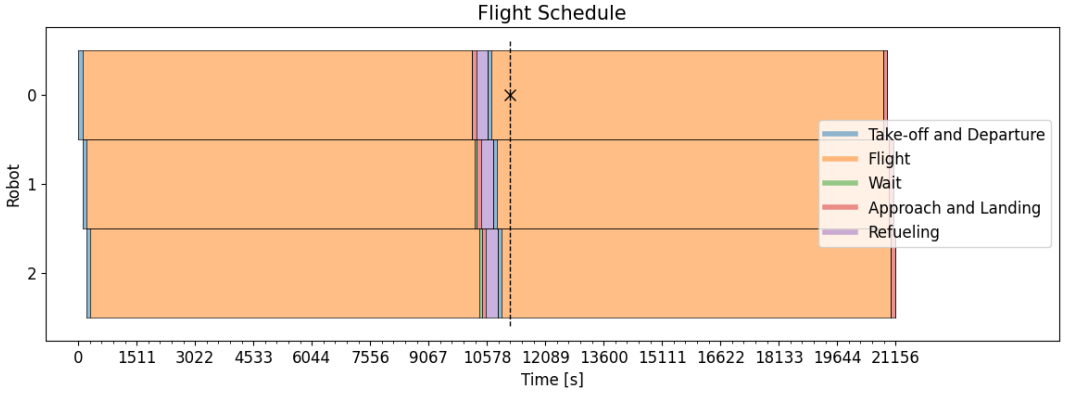


(a) Coverage paths for Aberdeen example environment with a central ground station.

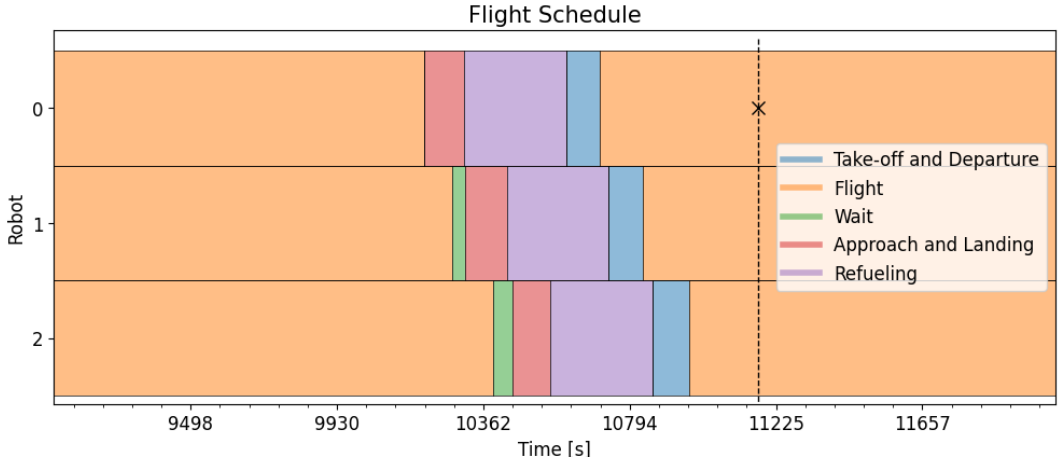


(b) Snapshot of the Aberdeen example environment at point of target detection

Figure 8.21: Aberdeen example environment with three UAVs that refuel.



(a)



(b)

Figure 8.22: Flight schedule for the Aberdeen example environment with three UAVs that refuel.

Robot	Refuel	Take-off [min]	Departure [min]	Flight [min]	Wait [min]	Approach [min]	Landing [min]	Distance [km]	Rotations
0	0	0.8	1.0	168.0	0.0	0.9	1.1	141.1	166.0
0	1	0.8	0.9	168.8	0.0	0.7	1.1	141.8	158.0
1	0	0.8	0.9	167.8	0.6	0.9	1.1	140.9	204.0
1	1	0.8	0.9	169.3	0.0	0.7	1.1	142.2	134.0
2	0	0.8	0.9	168.1	1.0	0.7	1.1	141.2	188.0
2	1	0.8	1.0	167.7	0.0	0.9	1.1	140.9	206.0

(a)

Robot	Refuel	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]
0	0	171.8	178.8	360.0	181.2
0	1	172.2	178.9	360.0	181.1
1	0	172.1	180.6	360.0	179.4
1	1	172.8	178.6	360.0	181.4
2	0	172.6	180.6	360.0	179.4
2	1	171.6	180.1	360.0	179.9

(b)

Figure 8.23: Data tables generated for the Aberdeen example environment with three UAVs that refuel.

5 hours and 50 minutes to complete. The target is found quite early into the second sub-region search by robot 0. The time of target detection is about 3 hours and 6 minutes into the search. Robot 0 covers the blue sub-regions, where the target is located, of Figure 8.21b.

Jeffreys Bay

The Jeffreys Bay example environment is also explored. It is a small environment and it is assumed that only one GULL UAV is available for the search. The constant search speed in this scenario is 19 m/s. Normally, these UAVs have a long endurance. It is assumed in this case that a large portion of the UAV payload capacity is not available. It may carry supplies for survivors or perhaps additional electronic equipment. It is assumed the UAV has a 30 minutes flying time available. In that case, it needs to refuel once to complete its search of this region. Since there is only one UAV, the regions of Figures 8.25a and 8.21b are both shades of blue. The target location for Figure 8.25b is at 2 km on the x-axis and 1.1 km on the y-axis. This figure shows a snapshot at the time of detection.

In this case the full flight plan would take about 43 minutes and target detection occurs at roughly 36 minutes into the search. This detection time is indicated in Figure 8.26.

Detection occurs after having refuelled once. The table in Figure 8.24a shows the times associated with the flight plan. No wait times are induced since there is only one UAV. The table in Figure 8.24b indicates a feasible flight plan since the excess is once again positive.

A 30 minute flight limit is not uncommon for multi-rotor UAVs. This implementation could then also be seen as a demonstration of how a plan could look for such a vehicle, and how they are only suitable for quite small areas.

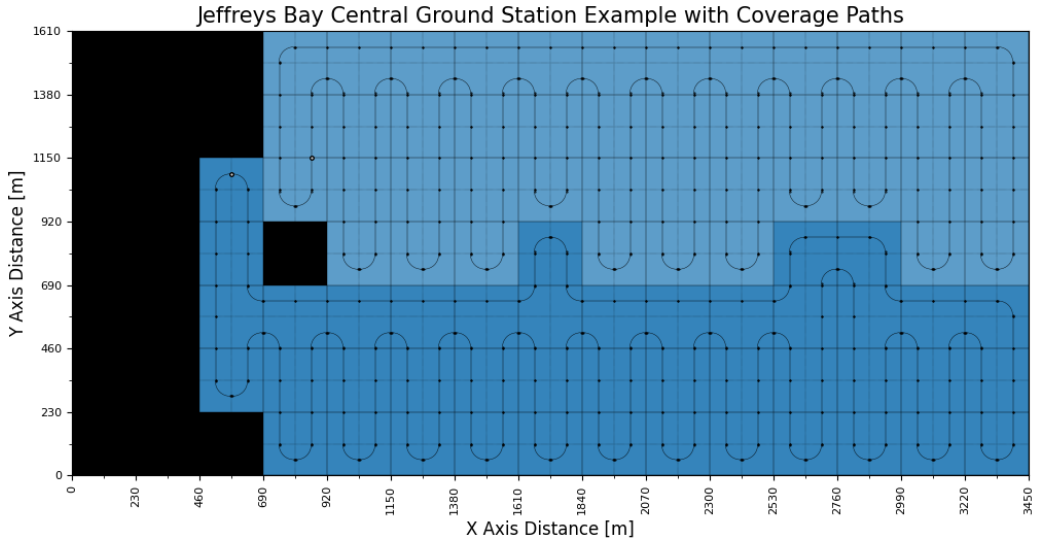
Robot	Refuel	Take-off [min]	Departure [min]	Flight [min]	Wait [min]	Approach [min]	Landing [min]	Distance [km]	Rotations
0	0	0.7	0.4	16.1	0.0	0.3	1.2	18.4	62.0
0	1	0.7	0.5	16.8	0.0	0.3	1.2	19.2	48.0

(a)

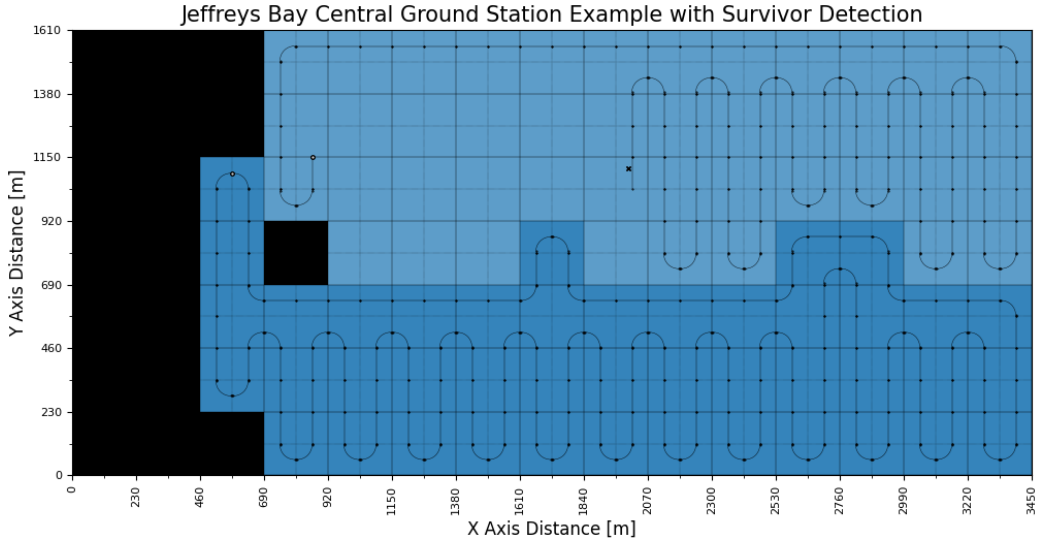
Robot	Refuel	Total Time [min]	Total Energy [min]	Time Limit [min]	Time Excess [min]
0	0	18.8	21.1	30.0	8.9
0	1	19.5	21.4	30.0	8.6

(b)

Figure 8.24: Data tables generated for the Jeffreys Bay example environment with one UAV that refuels.



(a) Coverage paths for Jeffreys Bay example environment with a central ground station.



(b) Snapshot of the Jeffreys Bay example environment at point of target detection

Figure 8.25: Jeffreys Bay example environment with one UAV that refuels.

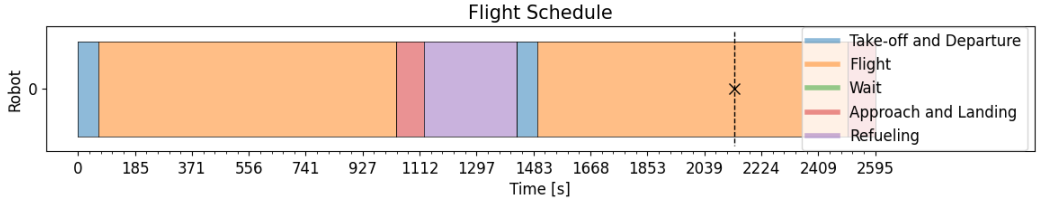


Figure 8.26: Flight schedule for the Jeffreys Bay example environment with one UAV that refuels.

Chapter 9

Monte Carlo Simulations

This chapter presents Monte Carlo simulation results for the automated search and rescue approach that was developed to use multiple unmanned aerial vehicles (UAVs) to cooperatively search a designated area. Section 9.2 presents simulation tests that were performed to investigated the execution times of the divide areas algorithm and the sub-region coverage algorithm. Section 9.3 presents simulations that were performed to test the survivor detection performance of the full system. The survivor detection performance was evaluated both in terms of the time taken to find the survivor, and the energy consumption of the UAVs. The survivor detection performance with and without the need for refuelling was evaluated and compared.

9.1 Experimental Setup, Procedure, and Results

9.1.1 Data Sets

The Monte Carlo simulations in this chapter were performed using two main data sets. The first data set contains a wide range of environment sizes, with a large variation in the environment shape, but with less simulation runs per environment size. The second data set contains four specific environment sizes with fixed environment dimensions, but with more simulation runs per environment size. The first data set contains a range of obstacle densities from 0% to 25% in increments of 5%. The second data set only contains two obstacle densities, namely no obstacles and 10% obstacle density.

Each data set is composed of three subsets. The first subset of simulations establishes the baseline for the algorithm and does not consider the operational constraints of the UAVs. The simulations are performed for environments with no obstacles and random UAV initial positions, with different environment sizes and different numbers of UAVs. The endurance limits, flight scheduling, and central deployment are not incorporated. The second subset is the same

as the first subset, except that obstacles are introduced into the environment. The simulations are performed with the same environment sizes and numbers of UAVs as the first subset, but with a range of obstacle densities. The third subset of simulations incorporates central deployment, flight scheduling, and endurance considerations. These simulations are performed with two UAVs, but with the number of sub-regions increasing with the environment size, and assuming the UAVs will sequentially search the sub-regions with refuels in between.

Note that the first data set looks at a very wide range of environment sizes, with changing shapes. One environment size, which is represented as a number of cells, can have a near square shape or possibly be very elongated in either dimension. The effect of environment shape is not explicitly explored.

The changing shapes are introduced by sweeping through a number of cells between 10 and 100, in increments of 5, for both environment dimensions. A resulting 361 combinations are possible, ranging from a total environment size of 100 cells to 10000 cells. This is done for two and eight UAVs in the environment.

The second data set looks at four environment sizes explicitly. The shape is also kept constant in each case, with the vertical always being larger than the horizontal. The ratio between the number of rows and columns is kept between a value of one and two, to ensure the shape does not have a large impact on the results. The environment sizes vary between 2000 and 8000 total cells, in increments of 2000. The actual environment sizes used are 40x50 cells, 50x80 cells, 75x80 cells and 80x100 cells. For each combination of environment size and number of robots, 25 simulation runs are done. This ensures that the spread of the data can be explored, particularly for the divide areas algorithm where there is a random component to the data.

9.1.2 Simulation Setup and Procedure

A number of parameters are kept constant to make the results comparable. The maximum iterations, as a parameter for the DARP algorithm, is kept at 10000. Due to the nature of the algorithm this means that the actual maximum is closer to about 20000 iterations. The maximum allowable cell discrepancy is kept at 10% of the environment size. The discrepancy is not necessarily a function of environment size, but it is assumed that higher discrepancies are more acceptable with larger environments. This may not always be the case, particularly with high obstacle densities and more robots. For the environments tested in the simulation however, this gave an acceptable result. The DARP algorithm does seek to minimise the discrepancy, and so the limit is used as a last resort when finding a solution. The limit is rarely reached.

The weights of the random matrix and the connectivity multiplier in the DARP algorithm are also kept at a constant value. The connectivity weight is set to 0.1 and the weight of the random matrix is kept at 0.001. The trend for

these values is that higher values lead to less iterations and lower execution times. Higher values, however, increase the number of failed runs. The random component should generally be lower than the connectivity component to achieve success. There are cases where the maximum iterations are reached before a solution is found. A detailed investigation of the relationship between the algorithm results and the weights is however not performed.

All the simulations also make use of one assumed UAV and camera combination. The Strix 400 UAV is used in combination with the Sony RX1R II camera. The ground is assumed to be level, so the take off and landing heights are equal to the ground elevation. The height of the UAV above ground is assumed to be 160 m; it is assumed to fly at its cruise speed of 14 m/s; and the time taken to refuel this UAV is assumed to be five minutes. With these values, the small cells in the environment have a roughly 90 m dimension. The environments tested therefore range in size from roughly 3 km^2 to 320 km^2 in the first data set. This simulation therefore addresses a large variety of environments.

The Monte Carlo simulations were run on the same device. The components include an AMD Ryzen 7 5800H processor with a clock speed of 3.2 GHz; 16 GB DDR4 RAM with a RAM speed of 3200 MHz; and a 512 GB SSD.

9.1.3 Simulation Failures

With all the simulations that are run, there are occasionally runs that are aborted. In the case with no obstacles, this means no solution was found for the divide areas algorithm. There are a few edge cases where a solution may not be possible, for example when the UAVs are tightly clustered and block each other.

Aside from that, it would mean a solution exists but it can simply not be found within the maximum number of iterations allowed. This maximum can be increased, or the weights of the random and connectivity component can be changed in order to find the solution. As of yet, no framework is available for selecting weights appropriate to a specific scenario. It would be valuable if one was developed, but this is beyond the scope of this project.

In the case where there are obstacles, it is also possible that during random obstacle generation, enclosed regions are formed. These enclosed regions are viewed as obstacles, in this implementation of the algorithm, which increases the obstacle density. If it is increased by more than 1%, the algorithm is aborted. Similarly, if a robot is removed in the process, the algorithm is aborted.

In these simulations, aborted runs are rerun until there is a successful equivalent. Results are only shown for those that successfully find a solution.

9.1.4 Simulation Results

Section 9.2 presents simulation tests that were performed to investigated the execution times of the divide areas algorithm and the spanning tree coverage algorithm. This includes performance measures such as execution time and number of iterations. The sub-region coverage technique and the divide-areas algorithm are looked at separately. The performance of the divide areas algorithm in terms of discrepancy is shown. This indicates how well it divides the area. Algorithm failures are also briefly explored. These failures do not include aborts due to obstacle density changes or robot removals.

Section 9.3 presents simulations that were performed to test the survivor detection performance of the full system. The predicted time for path completion is shown, which gives an indication of the longest time it would take to find a target. The estimated energy consumption is also shown to show the efficacy of the refuelling protocol.

Lastly, the Aberdeen environment is explored specifically. This shows the variation in survivor detection times one can expect in a single environment. Two separate simulations were run for this scenario, with constant values more appropriate to this case specifically. Section 9.3.2 discusses this further.

9.2 Algorithm Execution Time

In this section, the algorithm performance is discussed largely in the sense of execution time and number of iterations to find a solution. The goal is to ascertain whether the algorithm can feasibly be used in a time sensitive SAR operation. Moreover, the general bottle-necks and possible improvements are discussed. Section 9.2.1 discusses the performance of the divide areas algorithm and Section 9.2.2 addresses the sub-region coverage technique. In both cases, central deployment and refuelling are investigated to assess its impact on the solution.

9.2.1 Divide Areas Algorithm Performance

This section presents the Monte Carlo simulation tests that were performed to investigate the execution time of the DARP algorithm that divides the designated search area into sub-regions. The execution time was measured both in terms of the total execution time of the algorithm, and the number of iterations taken to find a solution. Simulation experiments were performed to determine the execution time for different environment sizes, for different numbers of UAVs, for different obstacle densities, and finally for different deployment strategies (central deployment or distributed deployment). The discrepancy between the number of cells assigned to each sub-region, as well as the failure rate of the DARP algorithm, are also briefly discussed.

The Monte Carlo simulation results are mostly presented in the form of box-and-whisker plots. Because the DARP algorithm has a random component, it is appropriate to show the data in this format. The box-and-whiskers plots show the statistical distributions of the data, including the median and mean values, the 25^{th} and 75^{th} percentiles, and the minimum and maximum values.

The number of iterations and algorithm execution times were investigated using Monte Carlo simulations performed using the second data set described in Section 9.1. The second data set uses four specific environment sizes with fixed environment dimensions. More simulation runs are done per environment size, making it more appropriate when investigating the spread of the data. The second data set only contains two obstacle densities, namely no obstacles and 10% obstacles.

DARP Execution Time vs Environment Size and Number of UAVs, No Obstacles

Simulations tests were performed to investigate the effect of the environment size and the number of UAVs on the number of DARP iterations and the total DARP execution time. The first set of simulations was performed with environments that contain no obstacles.

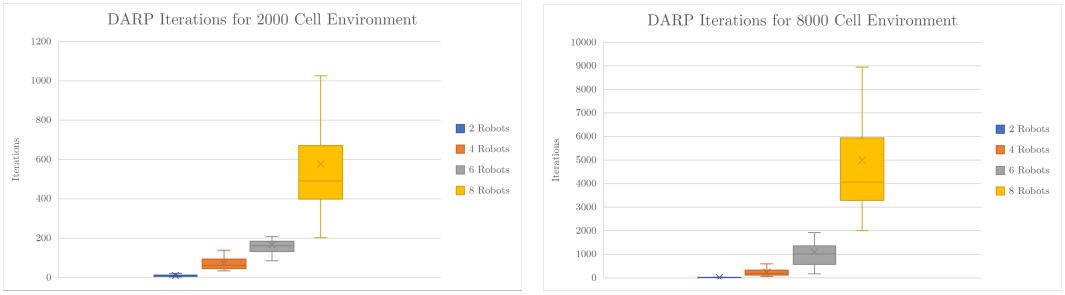
Monte Carlo simulations were performed with four different environments sizes, ranging from 2000 to 8000 cells in increments of 2000 cells, to show the effect of environment size on the algorithm execution time. (This is the number of large cells, with each large cell consisting of four small cells.)

Assuming a flying height of 160 m, the small cell discretisation size is 89.8 m, and the large cell dimension is then 179.5 m. This means that a 40x50 cell environment (2000 cells total) represents a 7.1 km by 9 km search area. A 80x100 cell environment (8000 cells total) represents a 14.4 km by 18 km search area.

Figures 9.1a and 9.1b show the distributions of the number of DARP iterations as a function of the number of UAVs for the smallest (2000 cell) and largest (8000 cell) environments, respectively. Figures 9.2a and 9.2b show the corresponding distributions of the total DARP execution times as a function of the number of UAVs.

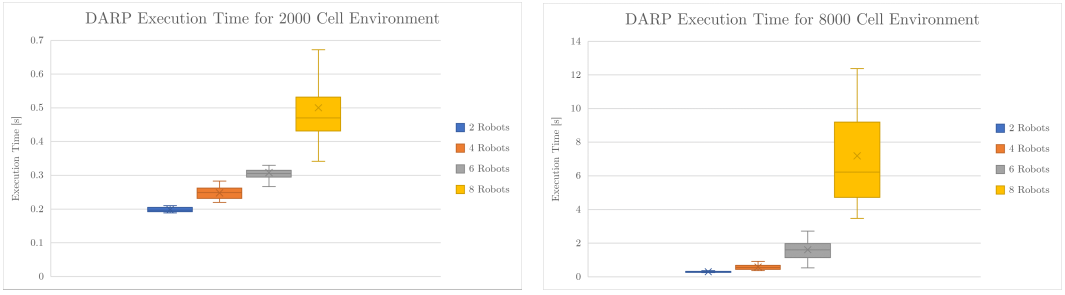
The box and whisker plots show the median, the 25^{th} and 75^{th} percentiles, and the minimum and maximum values of each distribution. In addition, the "**X**" marks the mean value.

The simulation results show that the number of DARP iterations and the algorithm execution time generally increase with the environment size and the number of UAVs. More UAVs means that more sub-regions must be formed from the same environment. There are more boundaries that need to be negotiated between the regions to form contiguous, equal-sized sub-regions.



(a) 40x50 Cell Environments (b) 80x100 Cell Environments

Figure 9.1: Box-and-whisker plots of DARP iterations for simulations in no obstacle environments containing UAVs with random initial positions.



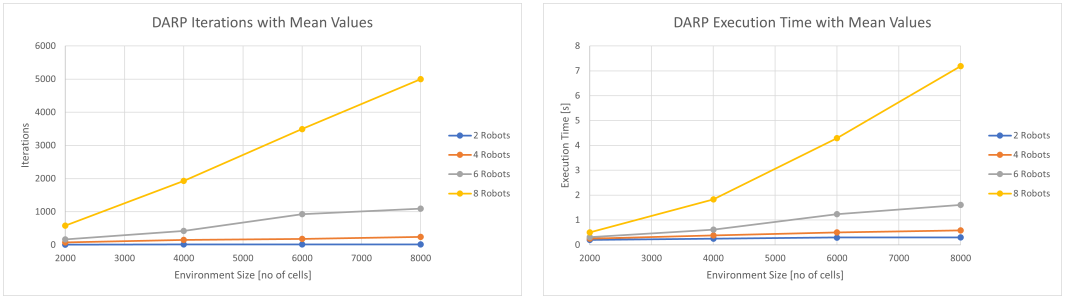
(a) 40x50 Cell Environments (b) 80x100 Cell Environments

Figure 9.2: Box-and-whisker plots of DARP execution time for simulations in no obstacle environments containing UAVs with random initial positions.

The number of iterations and execution times also vary more as the number of UAVs increases. The likely reason for this is that the range of possible UAVs configurations increase with an increase in the number of UAVs. There are a wider range of possibilities which can make the algorithm execute faster or slower. It is likely that more closely clustered UAVs would cause the algorithm to take longer to execute.

It is also clear from the results that an increase in environment size causes an increase in both the number of iterations and the execution time of the algorithm. This is evident by looking at the mean values in particular. On average it takes more iterations and therefore more time to negotiate assignments when there are more cells.

To show the trend more clearly, the mean values for the four different environment sizes are plotted in Figure 9.3. The DARP iterations and the execution time increase nearly linearly with the increase in the environment size.



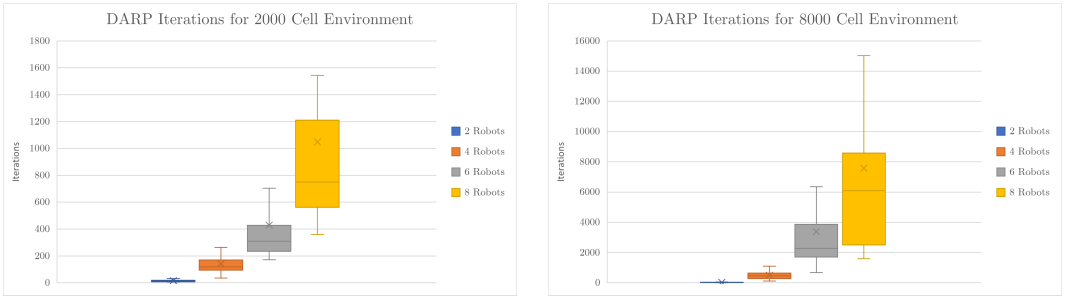
(a) Iterations (b) Execution Time

Figure 9.3: Mean values of DARP algorithm for simulations in no obstacle environments containing UAVs with random initial positions.

DARP Execution Time vs Environment Size and Number of UAVs, for 10% Obstacle Density

A second simulation was run for environments with a 10% obstacle density. These simulations were performed using the same experimental setup as for the environments with no obstacles, except that 10% of the free cells were randomly marked as obstacles.

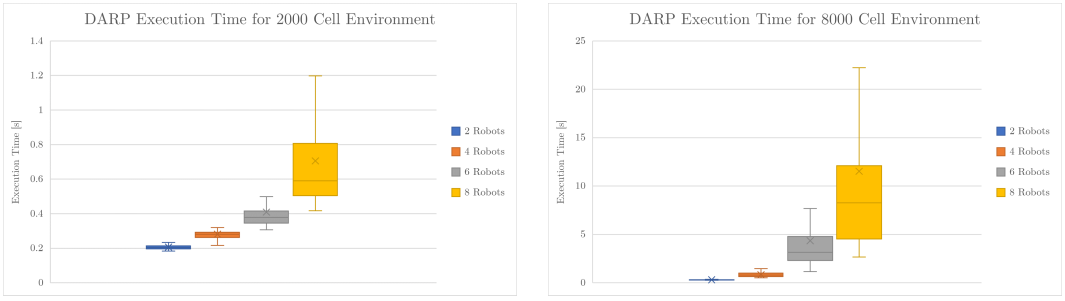
The Monte Carlo simulation results for the environments with 10% obstacle density are shown in Figures 9.4 and 9.5. Figures 9.4a and 9.4b show the distributions of the number of DARP iterations as a function of the number of UAVs for the smallest (2000 cell) and largest (8000 cell) environments, respectively. Figures 9.5a and 9.5b show the corresponding distributions of the total DARP execution times as a function of the number of UAVs.



(a) 40x50 Cell Environments (b) 80x100 Cell Environments

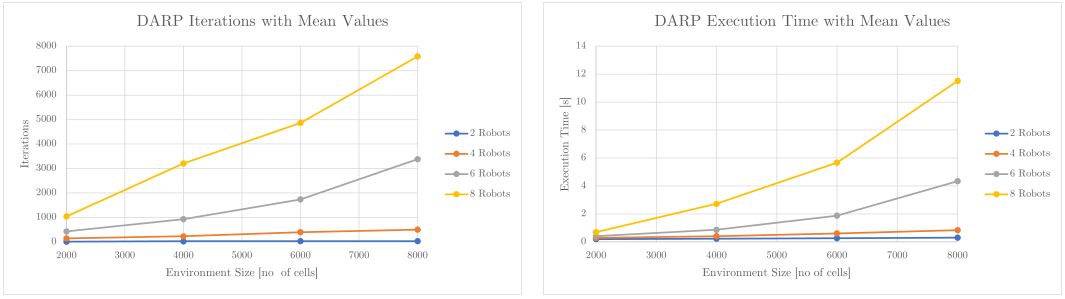
Figure 9.4: Box-and-whisker plots of DARP Iterations for simulations in environments containing 10% obstacles and UAVs with random initial positions.

As before, the iterations and execution time increase with increasing environment size and increasing number of UAVs. Once again, the mean values are plotted in Figure 9.6, showing the general linear behaviour with respect to environment size. The most important observation here is that the total number of iterations and the algorithm execution time are larger for the simulations



(a) 40x50 Cell Environments (b) 80x100 Cell Environments

Figure 9.5: Box-and-whisker plots of DARP Execution Time for simulations in environments containing 10% obstacles and UAVs with random initial positions.



(a) Iterations (b) Execution Time

Figure 9.6: Mean values of DARP algorithm for simulations in environments containing 10% obstacles and UAVs with random initial positions.

with 10% obstacle density compared to the simulations with no obstacles. The mean values for the number of iterations and algorithm execution time have increased for environments with 10% obstacle density relative to environments with no obstacles, despite the fact that the effective number of free cells has decreased by 10%.

It is clear that the algorithm takes longer to negotiate cell assignments when there are obstacles. This can be expected since the environment complexity has increased. UAVs may be forced to navigate narrow or isolated regions, but the algorithm still needs to negotiate assignments so as to form equal-sized, contiguous regions.

The variation in the resulting values also increases with respect to the no obstacle simulation. The reason for this would be because the obstacle configurations can vary greatly, causing more possible solutions. The no obstacle environment is a constant shape, and therefore solutions do not vary as greatly.

Effect of Central Deployment on DARP Execution Time

A final set of simulations was performed to look specifically at the effects of central deployment. The method used in this project configures the UAV initial positions around a central ground station which is viewed as an obstacle. The goal is to investigate the effect of these configurations on the algorithm.

For each simulation run, the ground station was given a random location in the environment. Aside from the ground station, no other obstacles were added to the environment. The UAV initial positions were then arranged in a perimeter configuration around the ground station. The simulations for the 2000 cell environment were performed assuming two UAVs. The simulations for the 8000 cell environment were performed assuming eight UAVs.

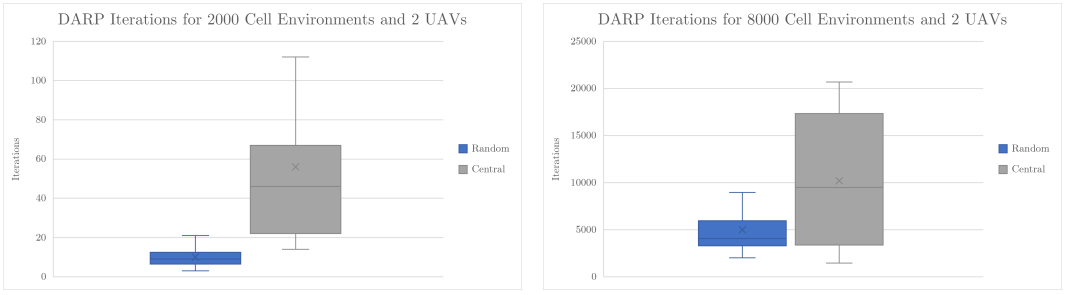
Figures 9.7 and 9.8 show the simulation results for central deployment compared to the simulation results for random UAV initial positions. The distributions for the central deployment are shown in grey, and the distributions for the random UAV initial positions are shown in blue. (The results for the random UAV initial positions were obtained from the first set of simulations.)

It is quite clear that the central deployment tends to lead to longer execution times and more iterations, even though the number of sub-regions do not change. There is also a greater variation in the data when compared with the random initial positions case. The closely clustered robot configuration and the ground station as an obstacle appear to make generating the sub-regions more challenging for the algorithm. This is a slight disadvantage of using the central deployment strategy.

Figure 9.9 shows a comparison of the mean values for central deployment to the mean values for random initial positions. The bar graphs show the average number of iterations and the average execution times for central deployment compared to random deployment, for four different environment size and number of UAV combinations. (The number of UAVs are increased proportionally to the environment size.) It is clear here that the central deployment causes a general increase in the number of iterations and the execution time for the divide areas algorithm.

The execution time of the divide areas algorithm is well under a minute for all of the Monte Carlo simulations that were performed, even for the largest environments with the largest number of UAVs. The execution time of the divide areas algorithm is therefore negligible compared to the time scales of a search and rescue operation. The maximum number of iterations is about 15000, which was observed for the largest environment with the largest number of UAVs, with a 10% obstacle density.

The number of iterations generally increase with increasing environment size, number of UAVs, and percentage obstacle density. Keeping this in mind, as these values are increased, it is likely the algorithm will start failing to find a solution within the maximum allowable iterations. The number of allowable iterations can then be increased if necessary, or the weights of the random and

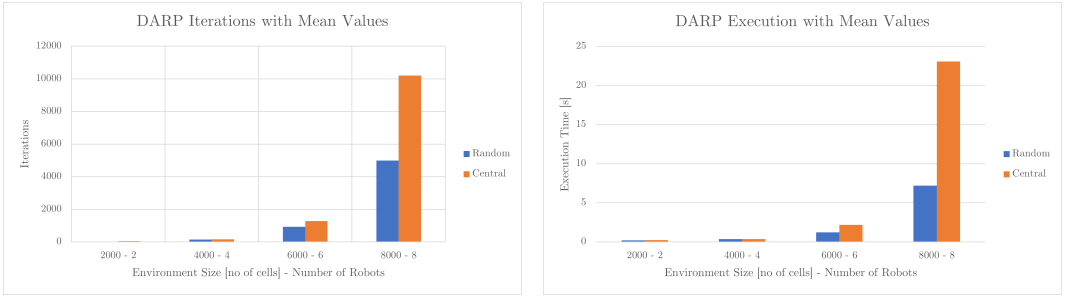


(a) 40x50 Cell Environments (b) 80x100 Cell Environments

Figure 9.7: Box-and-whisker plots of DARP Iterations for simulations in no obstacle environments comparing central deployment and random initial positions.

(a) 40x50 Cell Environments (b) 80x100 Cell Environments

Figure 9.8: Box-and-whisker plots of DARP Execution Time for simulations in no obstacle environments comparing central deployment and random initial positions.



(a) Iterations (b) Execution Time

Figure 9.9: Mean values of DARP algorithm for simulations in no obstacle environments comparing central deployment and random initial positions.

connectivity component can be altered. However, the effects of these weights will not be investigated in detail.

The low execution time is to be expected, since the divide areas algorithm was written in Java, which is a compiled language. Compiled languages tend

to have better runtimes.

Discrepancy value for number of cells between sub-regions

To evaluate the effectiveness of the divide areas algorithm to allocate equal-sized sub-regions to the individual UAVs, one can look at the discrepancy value. The discrepancy value is the difference between the maximum and minimum number of cells assigned to a sub-region. A discrepancy of one or zero indicates an essentially perfect area division. Anything higher than that indicates that the area division is imperfect. An imperfect division can occur due to environment shape. Regions of one cell width wide can make it impossible to get a perfect division with certain UAV initial positions. It is also possible that a better division can be achieved by increasing the maximum iterations or altering the weights of the random or connectivity components. The effects of altering these are not investigated.

The trends in the discrepancy value were investigated using Monte Carlo simulations performed using the first data set described in Section 9.1. The first data set contains a wide range of environment sizes, with a large variation in the environment shape, and a range of obstacle densities from 5% to 25% in increments of 5%. The discrepancy limit for the algorithm was made a percentage of the environment size. This is not because it is necessarily a function of the environment size, but because larger discrepancies are deemed more acceptable for larger environments for these simulations.

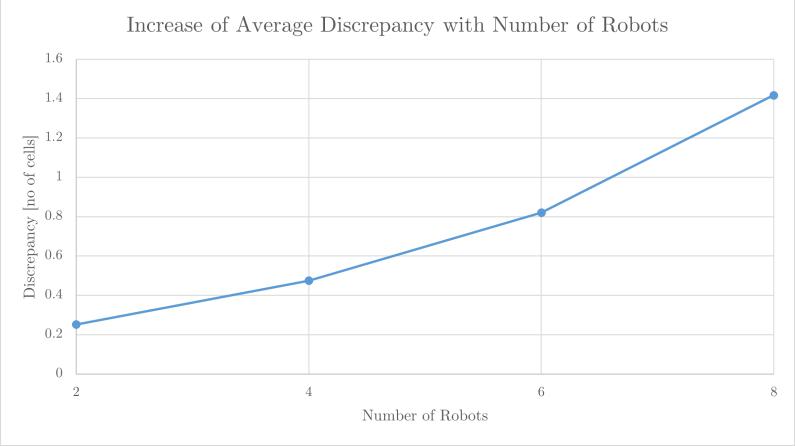
The Monte Carlo simulation results showed no correlation between the discrepancy value and the environment size. The discrepancy value does however appear to be linked to the number of UAVs, the number of refuels, and the obstacle density.

Figure 9.10 shows the the mean discrepancy value as a function of number of UAVs, as a function of obstacle density, and as a function of number of refuels (for two UAVs). Each mean value represents the average of the discrepancy values for environments ranging from 100 cells to 10000 cells. Since no correlation was found between environment size and discrepancy, the trends are clearly visible.

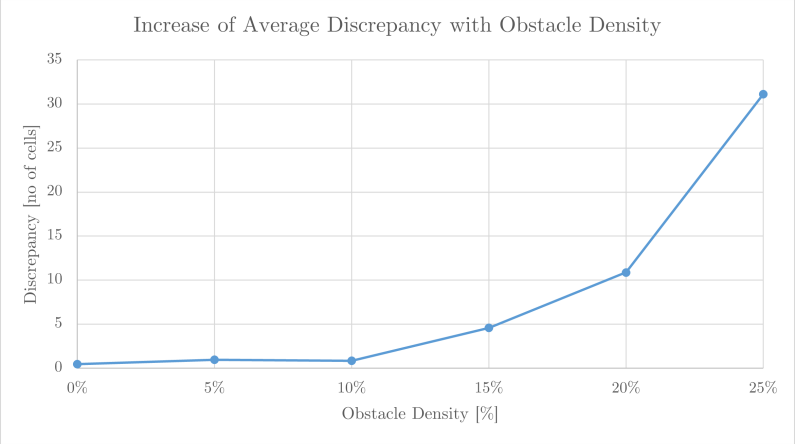
Figure 9.10a shows the results for environments with no obstacles and random UAV initial positions. There is a notable increase in discrepancy with the number of UAVs, but overall the divisions are still mostly perfect. The mean value barely exceeds a discrepancy of one, even with eight robots.

Figure 9.10b shows the results for an increasing obstacle density in the environment. The discrepancy value shows a clear, non-linear increase. The introduction of more obstacles causes more complex environments, leading to scenarios where perfect divisions are not possible. The more obstacles there are, the more of these edge-case scenarios are possible.

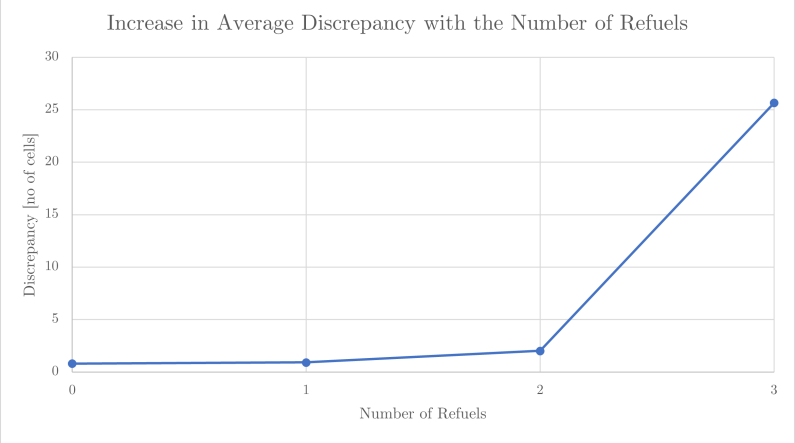
Figure 9.10c shows the results for environments with a central ground station, but no other obstacles. These simulation runs were performed using



(a) Discrepancy achieved in no obstacle environments with changing numbers of robots.



(b) Discrepancy achieved in environments with four UAVs and a changing obstacle density.



(c) Discrepancy achieved in environments with two UAVs and a changing number of refuels.

Figure 9.10: Graphs showing the average discrepancy achieved by the DARP algorithm in the simulations of the first data set.

two UAVs that refuel. Therefore, zero refuels corresponds to two equivalent UAVs. One refuels corresponds to four equivalent UAVs and so forth. Therefore, as with Figure 9.10a there is an increase in discrepancy with the number of UAVs/refuels.

It is interesting to note how the discrepancy reaches considerably higher values with the refuelling scenario, particularly for the eight equivalent UAVs case (three refuels per UAV). This is evidence that the perimeter configurations used for the refuelling protocol cause less perfect area division. This is likely because the UAVs are clustered closely together. With random UAV initial positions, there are likely only a few scenarios where they are all closely clustered together. With central deployment and the perimeter configuration, the UAVs are all adjacent to the central ground station obstacle as well, which may have an influence on the algorithm's performance.

DARP Failure Rate

The last performance measure that will be evaluated is the DARP failure rate. The simulations were set up to generate random environments. Therefore, UAV configuration and obstacle complexity is not predictable. The algorithm can occasionally not solve an environment before reaching the maximum iterations.

If the algorithm fails, the simulation is set up to rerun. It runs an environment with the same number of UAVs and obstacles, but likely in a new configuration. The graph in Figure 9.11 shows the percentage of reruns with respect to the total number of runs in a simulation. These are for the four UAV case in the first data set, which encompasses a large range of environment sizes.

Obstacle density appears to be the main contributor to increased failures. Therefore, the relationship of failures to the environment size and the number of UAVs is not explored in detail for this project.

A rerun could be due to an unsolvable environment. The UAV and obstacle configuration could make a solution impossible. For example if there is a single cell width region that has a UAV initial position in it, this could cause problems. If a UAV is placed in the middle of several UAVs, there may also be no solution.

A rerun could also be because the maximum iterations are reached before the solution could be found. Changing the maximum iterations or the weights of the random and connectivity component would likely help find a solution.

Figure 9.11 clearly shows how the number of reruns increases with the obstacle density. With a 25% obstacle density, the number of reruns is nearly half the total runs. More obstacles cause reruns partially because there are more scenarios of unsolvable environments. However, it could also likely be that the environment complexity has increased and an increase in the maximum iterations or a change in weights would lead to a solution.

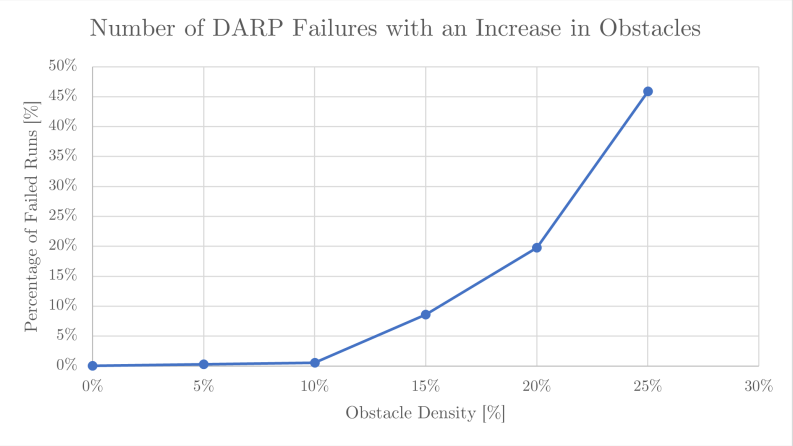


Figure 9.11: Graph showing the number of failures for increasing obstacle densities.

9.2.2 Sub-region Coverage Technique Performance

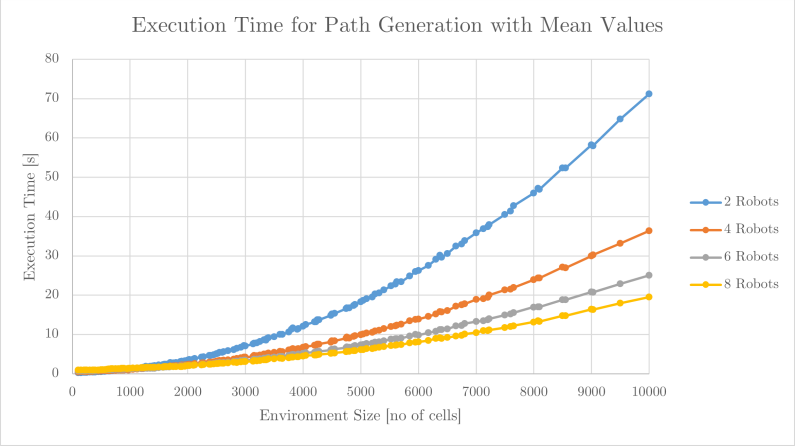
The sub-region coverage technique encompasses all the steps required to generate the final UAV coverage path including spanning tree generation, spanning tree circumnavigation, and path modification for dynamic constraints. This section presents the Monte Carlo simulation tests that were performed to investigate the execution time of the sub-region coverage algorithm that generates the sub-region coverage paths for each UAV. The execution time was measured in terms of the total execution time of the algorithm. An analysis was also performed to determine which components of the coverage path generation are the most time-consuming.

Simulation experiments were performed to measured the execution time for different sub-regions sizes, for different numbers of UAVs, for different obstacle densities, and for central deployment with UAVs that refuel.

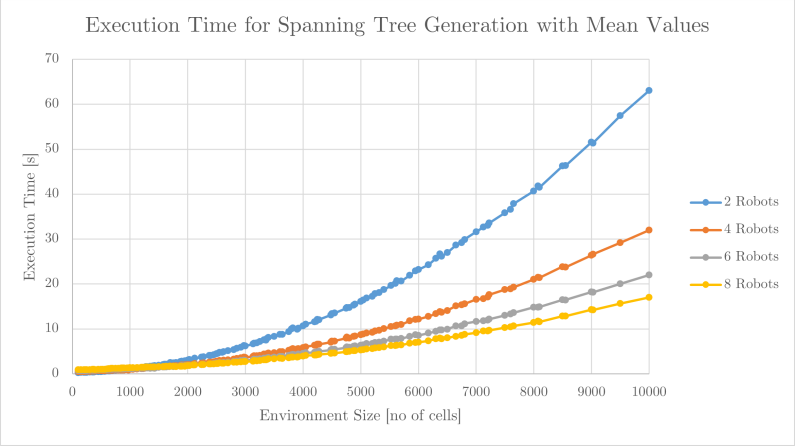
First, Monte Carlo simulations were performed using the first data set described in Section 9.1, with the larger variety of environment sizes. The simulations were performed for the environment with no obstacles, and for random UAV initial positions.

Figure 9.12 shows the mean execution time of the sub-region coverage algorithm as a function of the number of free cells in the environment. Three graphs are shown to show the contribution of each step of the coverage path generation to the overall execution time. Figure 9.12a shows the total time it takes to generate the coverage path. This includes spanning tree generation, tree circumnavigation, and the addition of dynamic constraints. Figure 9.12b isolates only the time taken to generate the spanning tree. Figure 9.12c isolates the time taken to generate the path that circumnavigates the spanning tree and to add the dynamic constraints. This is also called the waypoint generation time.

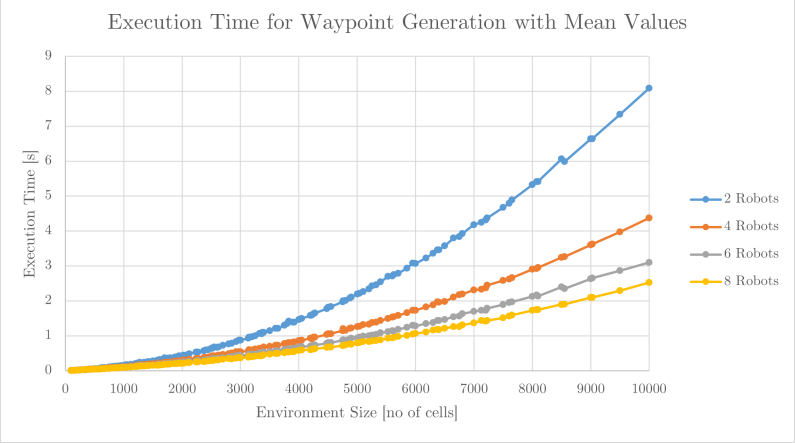
It is clear that generating the spanning tree takes the most time. This



(a) Execution time for coverage path generation



(b) Execution time for spanning tree generation



(c) Execution time for waypoint generation

Figure 9.12: Graph showing the time to generate coverage paths for no obstacle environments and random robot initial positions.

bottleneck was identified and is the reason that this portion of the code was written in a compiled language, namely Java. The rest was written in Python. With this in mind it is interesting to see that it still consumes the bulk of the time.

According to to X. Zeng et al. [57], the single robot spanning tree coverage solution runs in polynomial time. All three graphs here are non-linear in nature. This aligns with the original algorithm's behaviour, although here it is extended to the multiple robot case.

An increasing environment size causes a non-linear increase in execution time. However, even in the case of a 10000 cell environment, the path was generated in just over one minute. Combined with the DARP algorithm, the total algorithm runtime is in the order of a few minutes. This is a very reasonable runtime and is feasible for use in a SAR operation.

For the scenario with obstacles, there is expected to be little difference in the overall trends for execution time. This is provided the execution time is plotted with respect to the number of free cells in the environment, and not the full environment size.

The algorithm should be dependant on the number of cells per sub-region. However, when there is a larger discrepancy between cell assignments by the DARP algorithm, this would cause some variation.

Figure 9.10b in the previous section showed that the average discrepancy increases with obstacle density. The discrepancy implies an imperfect subdivision. The discrepancy for no-obstacle environments indicates a near perfect division in most of these scenarios. Therefore the data from the no-obstacle simulations can be used as a good baseline.

When the division is imperfect, the largest region will be slightly larger than a sub-region would be in an equally divided area. Therefore, the spanning tree will be larger and more waypoints would need to be generated. The overall result is that the spanning tree coverage technique would take slightly longer to execute for this region.

Figure 9.13 shows the mean execution time of the sub-region coverage algorithm as a function of the number of free cells in the environment, for different obstacle densities ranging from 0% to 25%. The dotted black line shows the baseline, which is the results from the no-obstacle simulation for four UAVs. The other data are from simulations with varying obstacle densities. The data follows the no-obstacle case very closely, with a general tendency to be slightly above the line.

Because some of the regions take slightly longer to execute, the overwhelming result is that the algorithm takes slightly longer to execute. The difference is very slight though.

Figure 9.14 shows the resulting execution times for path generation when refuelling and central deployment are incorporated. The dotted lines show the results for the no obstacle case, where the division is near perfect. The rest of the data shows the execution times as the number of refuels change.

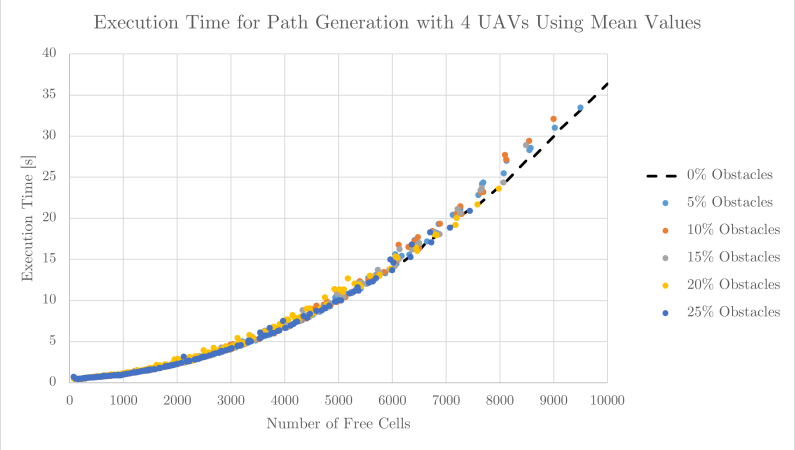


Figure 9.13: Graph showing the execution time of the sub-region coverage technique for four UAVs in environments of varying obstacle densities.

The simulations where refuelling was introduced were run with two available UAVs. Therefore, no refuels corresponds with two equivalent UAVs, two refuels correspond with four equivalent UAVs, and so forth.

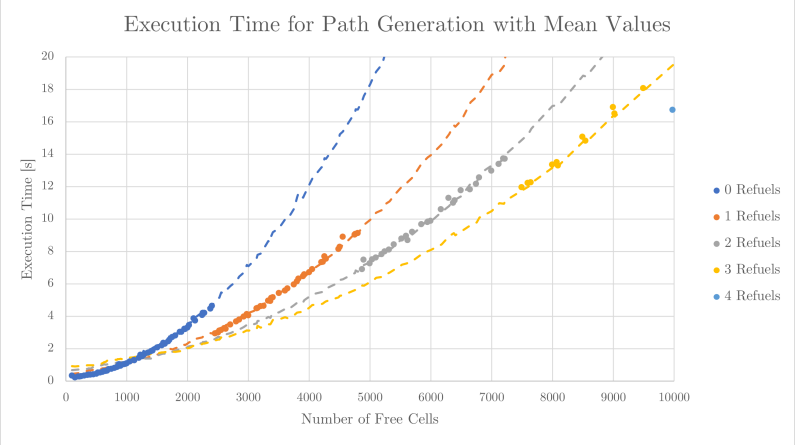


Figure 9.14: Graph showing the execution time of the sub-region coverage technique for two UAVs that refuel.

The blue dotted line is the results for two UAVs in no-obstacle environments, corresponding with two UAVs that do not refuel. The orange is for four UAVs, corresponding with two UAVs that refuel once. The rest is intuitive. The four refuel case is only one data point, and no simulation was run for ten equivalent UAVs to compare it with.

The simulation with obstacles and this refuelling simulation have very similar behaviour. The discrepancy for the refuelling case, as shown in Figure 9.10c, is also higher than the no-obstacle case on average. The data therefore tends to either follow the line or be slightly above it.

The data varies more as the number of refuels increase. This is to be expected since the discrepancies get higher with the number of refuels/UAVs. The data however follows the lines almost exactly and the effect discrepancy has here is minuscule. The perimeter configuration therefore does not cause a noticeable disadvantage for spanning tree execution time.

The divide-areas algorithm would behave the same for an environment that has four available UAVs, or with two available UAVs that refuel once. Either way the free cells need to be divided into four contiguous sub-regions. Therefore, it is expected that the refuelling data closely resembles the no-obstacle case with random UAV initial positions. Any variation would be due to the central deployment and perimeter configuration of the UAV.

From the previous figures, it is clear to see that the execution time decreases as the number of UAVs increase. However, to see this trend explicitly, the second data described in Section 9.1 set will be used.

Figure 9.15 shows the average execution times for different environment sizes as the number of robots change. The execution time decreases non-linearly as the number of robots increase. The reward of diminishing execution time is less as the robots are increased.

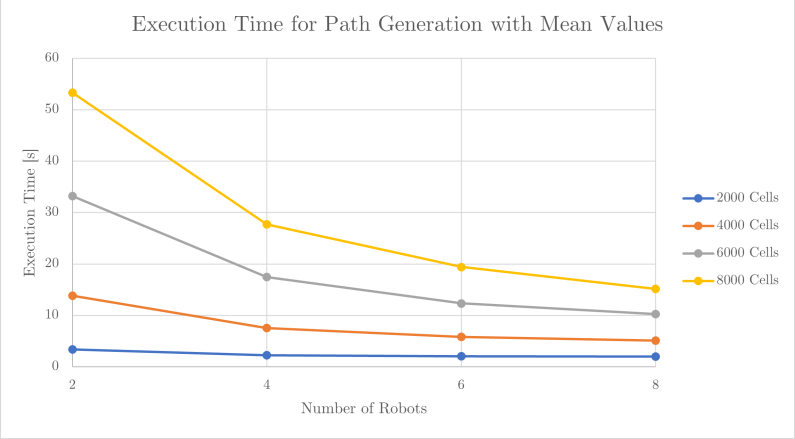


Figure 9.15: Graph showing the execution time of the sub-region coverage technique in no-obstacle environments as the number of robots increase.

Figure 9.16 shows the equivalent data for the case with 10% obstacles. The times on this figure are generally lower, since the number of free cells are effectively lower. The legend shows the overall environment size, not the number of free cells.

The individual sub-region sizes for the same environment are smaller when there are more robots. However, there are more total regions. The increase in execution time as a result of there being more sub-regions is less than the decrease in execution time as a result of the smaller sub-region sizes. This is

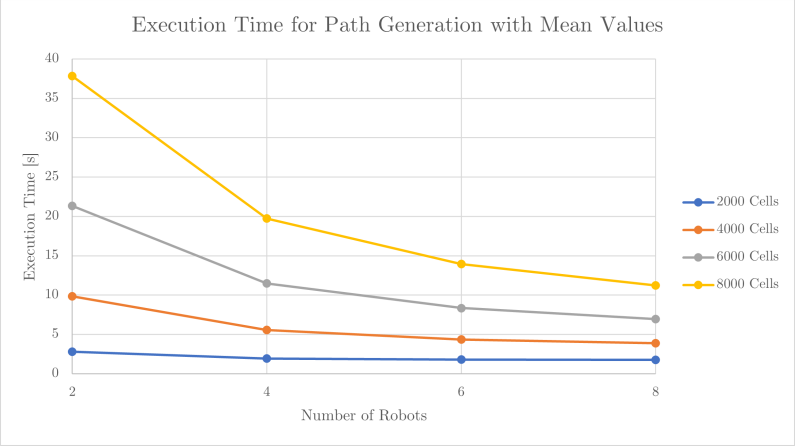


Figure 9.16: Graph showing the execution time of the sub-region coverage technique in environments with 10% obstacles as the number of robots increase.

likely because the spanning tree generation time increases non-linearly with sub-region size.

The effect of having more sub-regions seems to have more of an impact as the robots are increased. The trend could possibly reverse with enough robots in place, but this would likely be an impractical number of robots to use. The non-linear decrease is therefore sufficient to describe the overall behaviour of the algorithm.

9.3 Survivor Detection Performance

This section presents the Monte Carlo simulations that were performed to determine the time that it takes to locate a survivor. The survivor detection time was determined for different environment sizes, for different numbers of UAVs, and for different obstacle densities.

Simulation tests were also performed to investigate the effect of the UAV endurance on the survivor detection time. The survivor detection times for different numbers of UAVs that have limited endurance (i.e. they need to return to the central ground station to refuel) were compared to the detection times for different numbers of UAVs that have unlimited endurance (i.e. they do not need to refuel).

Section 9.3.1 presents the simulation tests that were performed to investigated the maximum time and the energy consumption to cover the entire search area. The schedule completion time is used as the metric for maximum search time, since it represents the maximum time it could take to locate a survivor in the search area. Section 9.3.2 presents the simulation tests that were performed to determine the distribution of survivor detection times for a specific environment, for different numbers of UAVs, and with and without refuelling. The Aberdeen environment was used as the test environment.

9.3.1 Maximum Search Time and Energy Consumption

This section presents the simulation tests that were performed to investigate the maximum time and the energy consumption to cover an entire search area.

The maximum time is representative of the maximum survivor detection time since this is the time the UAVs take to completely cover the search area. In an environment where the UAVs do not refuel, this is the maximum time taken by a UAV to cover a single sub-region. In most cases this would be the largest of the sub-regions. The maximum path completion time is equal to the time it takes to detect a survivor in the last cell that is searched by the UAVs. If the UAVs refuel, the maximum time becomes the duration of the generated flight schedule. The flight schedule duration is equivalent to the time the UAVs take to completely cover an environment. This, once again, is also the longest survivor detection would take.

The energy values in this section are also represented as a time. It is the time taken to search a sub-region, but with safety factors applied to account for more energy consumption during certain manoeuvres such as rotations. The maximum of these energy values should be below the predicted flight time of the UAV for the planned path to be feasible.

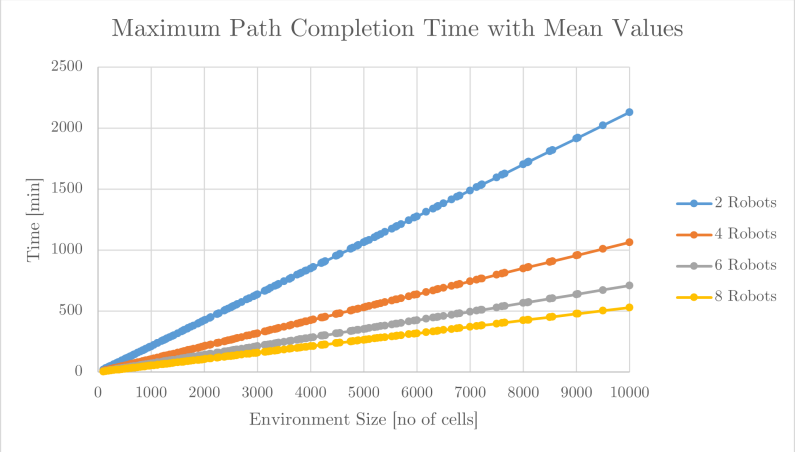
Survivor Detection Time vs Environment Size and Number of UAVs, with No Obstacles

The first set of simulations were performed for environments with no obstacles and random UAV initial positions, with different environment sizes and different numbers of UAVs. The endurance limits, flight scheduling, and central deployment were not incorporated. The simulations were performed using both the first and second data sets described in Section 9.1.

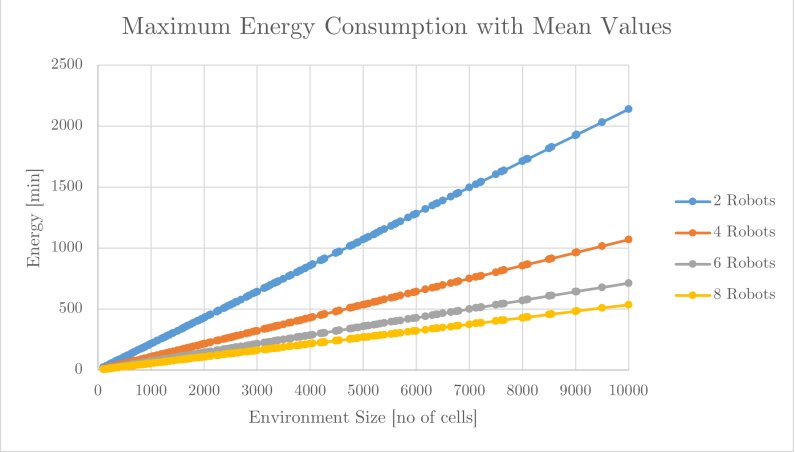
Figure 9.17 shows the simulation results for the first data set. Figures 9.17a and 9.17b show the maximum search time and energy consumption as a function of environment size and the number of UAVs. The results are intuitive and linear. Maximum survivor detection time increases linearly with environment size, because more cells take longer to cover. This figure also shows the results for different numbers of UAVs. More UAVs in general means a shorter search time, because the sub-regions are smaller. The maximum energy consumption result is practically identical to the maximum time result. The values are slightly larger but it is not noticeable. The flight limit of the Strix 400 is 9 hours, or 540 minutes. This limit is exceeded after a certain environment size. When the area is searched with two UAVs this happens at around 2500 cells.

The time results are important to consider in a search operation. The maximum time should not be impractical. For example, if a search starts in the morning, one would ideally want to complete it before dark. When using two UAVs, the longest search time is over 2000 minutes. This equates

to roughly 35 hours, which is likely impractical. By adding two UAVs, the maximum time halves to roughly 17 hours. It would be up to the search team to choose an appropriate number of UAVs to complete a search. If survivors are injured, the time constraint may be even shorter. Every time the number of UAVs in the search double, the search time essentially halves.



(a) Maximum path completion time.



(b) Maximum energy consumption.

Figure 9.17: Graphs showing the maximum time and energy consumed to completely cover environments of varying size with no obstacles.

Figure 9.18 shows the maximum search time as a function of the number of UAVs, for the four different environment sizes in the second data set. The results clearly show how the maximum search time halves every time the number of UAVs are doubled. The energy consumption is not shown here as it does not differ significantly from the time data.

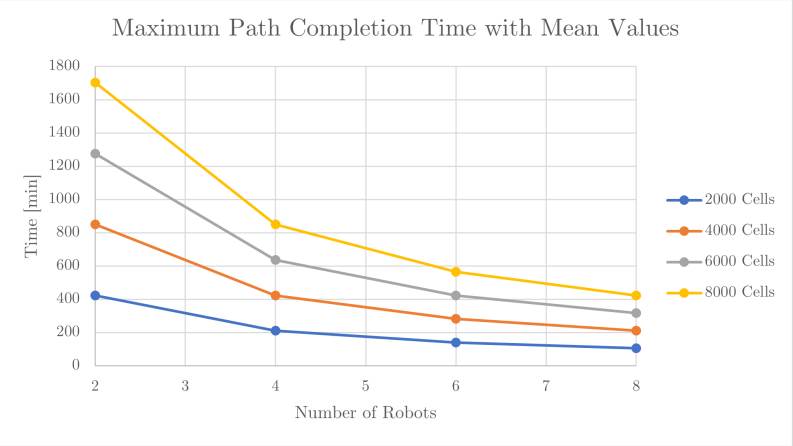


Figure 9.18: Graph showing the the maximum time to completely cover environments of varying size with no obstacles for an increasing number of UAVs.

Survivor Detection Time vs Environment Size and Obstacle Density

The second set of simulations were performed to investigate the effect of introducing obstacles into the environment. The simulations were performed using only the first data set described in Section 9.1. The simulations were performed with the same environment sizes and numbers of UAVs as in the previous section, but with obstacle densities ranging from 0% to 25%.

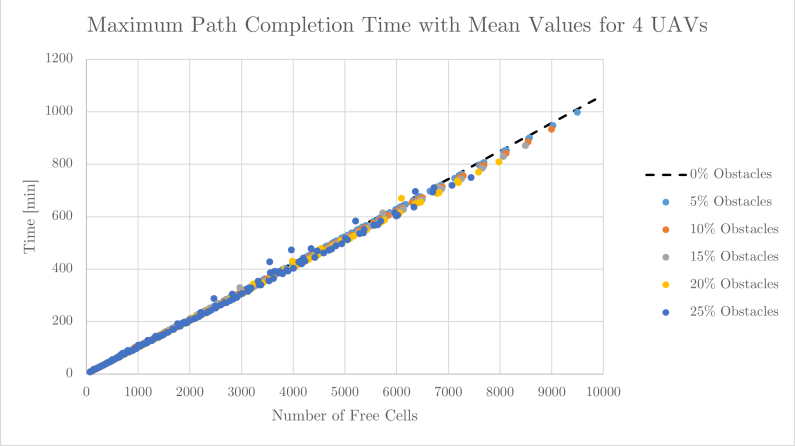
In Figure 9.10b it was shown that adding obstacles increases the average discrepancy in the divide areas algorithm. When taking only this into account, the maximum path completion is expected to increase slightly compared to the no-obstacles case. This is because a higher discrepancy means that the largest sub-region gets larger and the smallest sub-region gets smaller. Therefore it would take longer to finish covering the largest region.

Figure 9.19a shows the maximum search time as a function of environment size for various obstacle densities. The results assume that four UAVs were used to perform the search. The dotted black line represents the no obstacle data from the first data set. Although some of the data is above the line, likely due to the increased discrepancy, there is a significant portion of data below the line. The reason for this is that the number of rotations have increased. In this environment discretisation, rotations take less time to execute than straight-line manoeuvres. The increase in the number of rotations is expected with an increase in obstacle density. In more complex environments, the UAVs need to turn more often to achieve coverage. The values are plotted with respect to the number of free cells, so the environments are not decreasing in size with an increase in obstacles. The environment complexity is the only factor increasing the rotations.

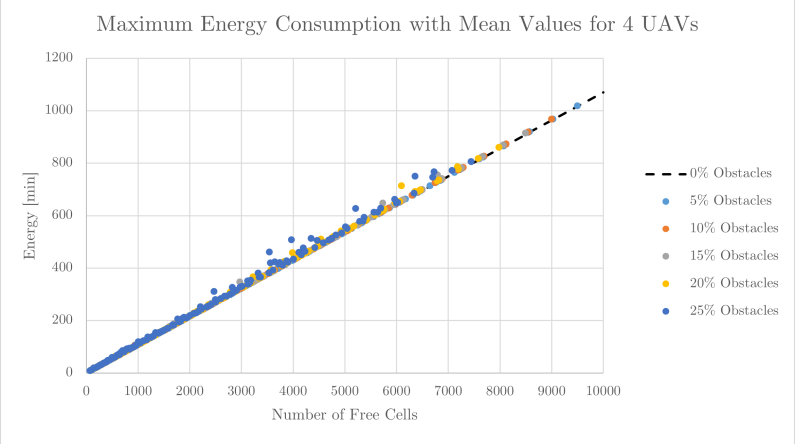
In general, obstacle density does not have a large effect on the overall trend of the search time versus environment size. However, the scale on the figure

is quite large, and there could actually be up to an hour difference between the search time in an environment with obstacles compared to an environment without obstacles. Depending on the scenario, this could be significant.

Figure 9.19b shows the energy consumption as a function of environment size for various obstacle densities. The results again assume that four UAVs were used to perform the search. Rotations take less time to execute, but a safety factor of 30% is applied for energy consumption. This means that turning manoeuvres effectively consume more energy than straight-line manoeuvres. Therefore, the energy consumption in an environment with obstacles is generally more than for an environment with no obstacles.



(a) Maximum path completion time.



(b) Maximum energy consumption.

Figure 9.19: Graphs showing the maximum time and energy consumed to completely cover environments of varying size and obstacle density.

Survivor Detection Time with Central Deployment and Refuelling

The third set of simulations were performed to investigate the effect of central deployment and refuelling The simulations were performed using only the first data set described in Section 9.1. These simulations are performed with two UAVs, but with the number of sub-regions increasing with the environment size, and assuming the UAVs will sequentially search the sub-regions with refuels in between. A flight schedule is also generated, with the times taken to take off, depart, search, arrive, land, and refuel all taken into account.

Figure 9.20a shows the maximum search time as a function of environment size, when using two UAVs with limited endurance that refuel between flights. The maximum search time now represents the total time to complete the flight schedule, which represents the longest time it can take to detect survivors. The maximum search time is plotted against the number of free cells in the environment, for consistency.

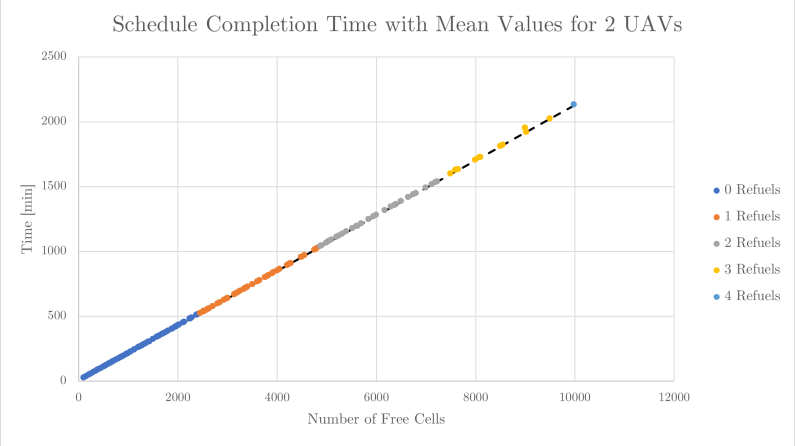
The black dotted line is taken from the no-obstacle simulation for two UAV where there is no refuelling and no central deployment and serves as a baseline for comparison. For all the refuelling cases, the line is followed quite closely. There is slightly more variation as the number of refuels increases, but not noticeably so. The additional manoeuvres for refuelling therefore barely affect the survivor detection time.

Figure 9.20b shows the corresponding energy consumption as a function of environment size, when using two UAVs with limited endurance that refuel between flights. The black line shows the 540 minute cutoff, equating to the 9 hour predicted flight time. The conservative approach for calculating the number of refuels is effective since the energy consumption estimation generally does not exceed 520 minutes.

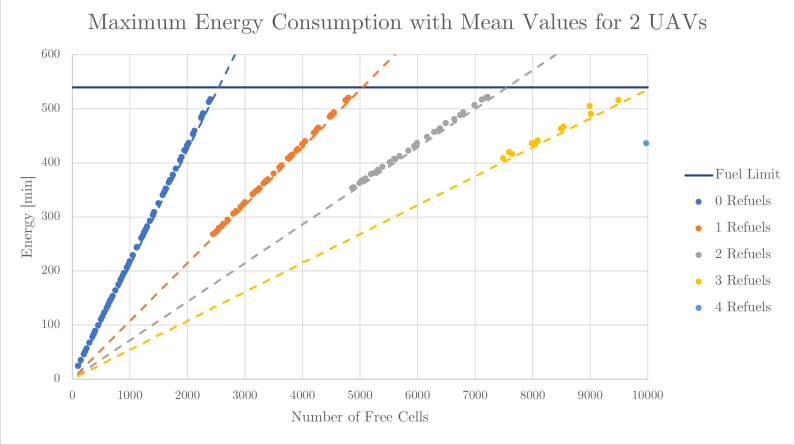
When looking at the total data, and not the average values, there were only two failure cases where the cutoff time was exceeded. The failure rate in this data was therefore 0.1%. These were also the only edge cases where the algorithm came close to reaching the 10% discrepancy limit. Changing the algorithm parameters and rerunning these scenarios could be used to remedy this. Decreasing the discrepancy limit may also be a good course of action, based on how it increases the flight times.

The dotted lines on the figure show the data from the corresponding no-obstacle simulation. The zero-refuels data follows the two UAV trend. The one refuel case follows the four UAV trend and so on. This is intuitive since one refuel with two UAVs leads to four equivalent robots. Therefore there are four sub-regions, as with the no obstacle and no refuel case with four available UAVs.

Generally, the energy consumption for the refuelling case is higher than with the no-obstacle case. This is likely due to the safety factor that is applied to the additional manoeuvres outside of the flight time. The ground station size also increases for the six and eight robots case, compared to the two and



(a) Maximum path completion time.



(b) Maximum energy consumption.

Figure 9.20: Graphs showing the maximum time and energy consumed to completely cover environments of varying size with no obstacles and UAVs that refuel.

four robots case. The approach and landing manoeuvres consequently become lengthier for these scenarios. This is why the increase in energy consumption is more significant for the corresponding two and three refuels case, compared to the zero and one refuels case.

The results from this section can be compared to the complete SAR solution that was developed by DroneSAR [1]. They claim that a search team took two hours to find a target in an area on 1 km^2. Comparably, a single quadrotor UAV took 20 min using back-and-forth manoeuvres.

The smallest area tested with the technique developed in this report was about 3 km^2 in size. A zoomed-in view of the data from Figure 9.20a is shown in Figure 9.21. The maximum survivor detection time for an area of this size was found to be roughly 30 min with two UAV. This is not a direct comparison, but it is an acceptable result, especially when compared with the time it took

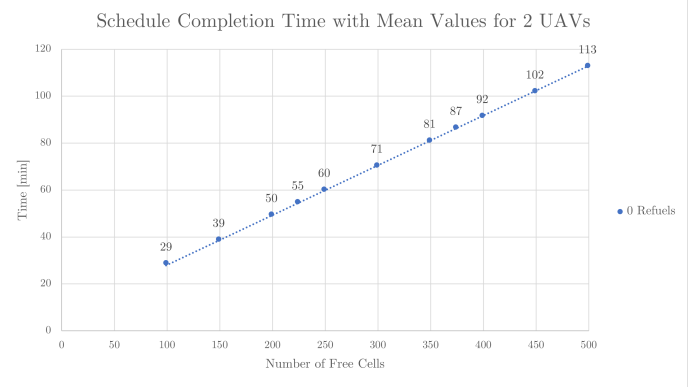


Figure 9.21: Zoomed in view of the maximum time to completely cover environments of varying size with two UAVs that are deployed from a central ground station.

a SAR team to find the target manually in a smaller area.

9.3.2 Survivor Detection Time in a Specific Environment

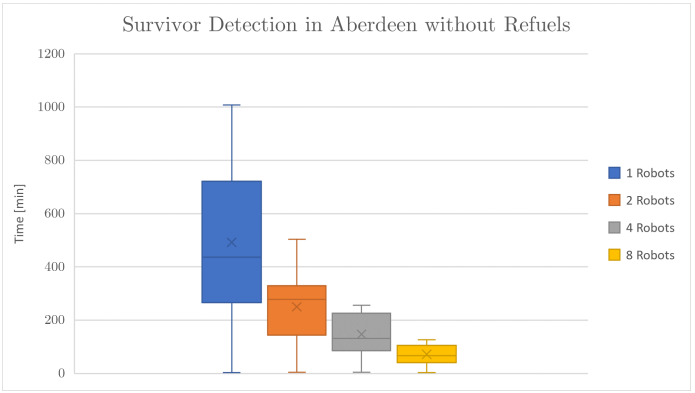
This section presents the simulation tests that were performed to determine the distribution of survivor detection times for a specific environment, for different numbers of UAVs, and with and without refuelling. The Aberdeen environment was used as the test environment. Survivor detection in this section is the time that has elapsed since the first UAV takes off until the target is found.

The simulations were all performed using central deployment, with the initial UAV positions arranged around the perimeter of the ground station. Take-off and landing times were also taken into account for the survivor detection time. The Strix 400 UAV was used, but its endurance was varied to demonstrate the effects of refuelling.

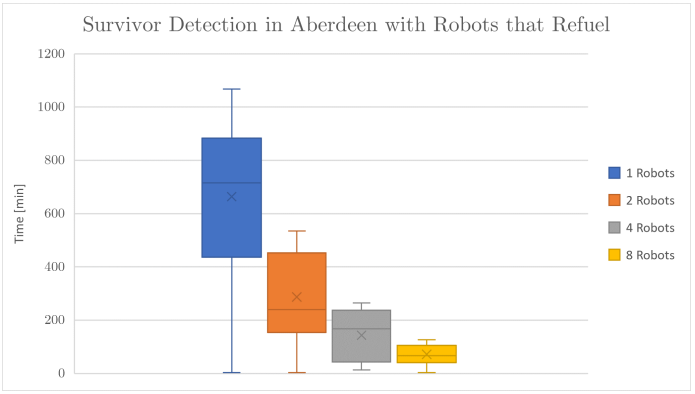
Exactly 25 different simulation runs were performed. Each simulation run used a different random ground station location and random unknown survivor location. Both the ground station location and the actual survivor location were drawn from a uniform probability distribution over the entire environment. The survivor detection times are based on the actual target location, and are not the maximum search times shown in the previous section.

The simulations were first performed for a changing number of UAVs that do not refuel. They are assumed to have enough fuel to complete their search without refuelling. Therefore, the endurance limit is different in each case.

Figure 9.22a shows the distribution of the survivor detection times as a function of the number of UAVs, assuming that the UAVs have unlimited endurance and do not need to refuel. The distribution is shown using box-and-whiskers plots that represent the median time, the 25^{th} and 75^{th} percentiles,



(a) Robots do not refuel.



(b) Robots refuel (eight equivalent robots).

Figure 9.22: Box-and-whisker plots of survivor detection time for 25 scenarios in the Aberdeen environment with a changing number of available robots.

and the minimum and maximum times. The "**X**" marks the mean survivor detection time. The simulation results clearly show that increasing the number of UAVs reduces the survivor detection time. Therefore, a search team will always benefit, in terms of survivor detection time, if they are able to utilise more UAVs.

The same simulations were repeated, but this time assuming that the UAVs have limited endurance, and need to refuel between covering sub-regions. The endurance limit was chosen so that a single UAV could only cover an eighth of the entire search area in a single coverage flight. This means that a single UAV would have to perform eight flights with seven refuels, or two UAVs would have to perform four flights each with three refuels each, or eight UAVs would have to perform one flight each, with no refuels.

The results show that the survivor detection times for UAVs with limited endurance are very similar to the survivor detection times for UAVs with unlimited endurance, when the same number of physical UAVs are used to perform the search. The survivor detection times are slightly higher for UAVs with limited endurance, due to the additional time required to approach, land,

refuel, take off, and depart. For eight UAVs, the survivor detection times are identical with and without limited endurance, because the UAVs with limited endurance each only perform one flight and do not need to refuel.

9.4 Key Findings

The execution time of the divide-areas algorithm was tested in simulation. The mean execution time of the DARP algorithm was found to increase with an increase in environment size and with an increase in UAVs. Introducing obstacles into the environment also caused an increase in the average execution time of the algorithm, due to an increase in environment complexity. When looking at obstacle free environments, it was shown that adding central deployment and perimeter configurations increased the average execution time compared to environments with random UAV initial positions.

To quantify how well the divide-areas algorithm divides a search area into equal-sized sub-regions, the discrepancy value was used. The area-division discrepancy was higher for environments with higher obstacle densities. This can, once again, be attributed to an increase in environment complexity with an increase in obstacle density. When looking at environments with no obstacles, those using perimeter configurations showed higher discrepancies than those with random UAV initial positions, but the difference was slight. In general, discrepancy also increased with an increase in the number of UAVs.

The failure rate of the DARP algorithm was briefly discussed. An increase in environment obstacle density was found to increase the DARP failure rate. The DARP algorithm cannot distinguish between a failure to find a solution because a solution does not exist, and a failure to find a solution where one exists, but the parameters of the algorithm simply need to be changed to find the solution. A search team, however, may be able to visually identify environments where a perfect area division is not possible.

The spanning tree coverage (STC) algorithm execution time was also tested in simulation, and was found to increase with an increase in environment size, but decrease with an increase in the number of UAVs. Baseline values were generated in environments with no obstacles and randomly generated UAV initial positions. Introducing obstacles into the environment had little effect on this time. Central deployment and perimeter configurations also had no noticeable impact on the results. Slight deviations from the baseline were attributed to the discrepancy in area division as a result of introducing obstacles or using perimeter configurations and central deployment.

The execution time of the DARP algorithm was generally well under a minute. The execution time of the STC algorithm was also in the order of minutes. For the maximum case tested, where two robots were used in a 10000 cell environment, the STC algorithm took just over a minute to execute. The

execution time of both these algorithms is negligible when compared to the time scales of a search and rescue operation.

According to Oregon Health and Science University [80], it was found that 99% of people who were found alive were found within the first 51 hours of a SAR operation. The SAR operations that were used in this statistic were conducted throughout the United States (US) state of Oregon, where there is a varied landscape including mountains, deserts and rivers. This statistic shows that survivor detection time is critical to increase survival rates.

Simulations were performed for maximum survivor detection times and energy consumption during flight, for the automated SAR technique described in this report. These results are specifically for a Strix 400 UAVs flying at a constant speed of 14 m/s. The Strix 400 UAV has a 9 hour, or 540 minute, predicted flight time. When looking at the energy consumption results, the refuelling protocol had a near perfect success rate for keeping the maximum energy consumption below this limit.

The maximum survivor detection times increased linearly with an increase in environment size, and decreased with an increase in the number of available UAVs. Doubling the UAVs effectively halves the maximum survivor detection time for a specific environment size. For example, two robots searching a 10000 cell environment would take roughly 35 hours to cover the whole area, representing an area of roughly 320 km^2. Since this is more than a full day, this would likely be impractical. Eight robots conducting the same search would take less than 9 hours to cover the same area. The maximum times showed almost no change with the introduction of refuelling and central deployment. The perimeter configuration therefore keeps the departure and approach times of the UAVs relatively short, and prevents them from having a large impact on overall survivor detection times.

The simulations showed maximum search times that are feasible for use in real-world SAR operations, but it would be up to the search coordination team to decide best suites their needs. More UAVs generally improves the chances of finding survivors sooner, which would be necessary for time-sensitive operations.

The performance of the solution developed in this project was compared to the complete SAR solution that was developed by DroneSAR [1]. The results were comparable for when UAVs are utilized, and according to DroneSAR, this far outperforms a manual search team.

Chapter 10

Conclusions and Recommendations

This chapter summarises the research presented in this thesis, with reference to aspects of the research that performed well and aspects that could use improvement. Section 10.1 summarises the work done and Section 10.2 provides suggestions for potential future work related to this research.

10.1 Summary of Work Done

The main goal of this research was to develop an automated search and rescue (SAR) approach with multiple UAVs. Existing literature was reviewed regarding the SAR problem as well as existing implementations where UAVs were used to assist in SAR.

The automated SAR problem was formulated as a distributed, offline, multi-robot coverage path planning (MCPP) problem. Literature regarding coverage path planning (CPP) was reviewed for the single and multiple robot case. For MCPP, existing literature regarding the offline distributed, offline non-distributed and online case was investigated. Using multiple UAVs has the advantage of reducing the time to cover an environment. The distributed case gives the added advantage of eliminating collisions between UAVs.

The search and rescue problem and its components were modelled and conceptualised. These include the search area, the terrain, the UAVs, and the survivor(s) within the environment. It was found that a systematic, automated UAV search would likely search a subset of a larger search area that is demarcated by the search coordination team in the planning stage of a search.

For an aerial UAV search of a lowland, mountainous or marine environment, it was deemed reasonable to assume that the environment is known prior to the search. Therefore, an offline approach was used with static obstacles. The problem was also reduced to a two-dimensional one by assuming a constant altitude search. The target, or survivor(s), were assumed static for the

duration of the search and survivor detection was assumed to be implicit with downward-facing onboard cameras.

Complete coverage was made a requirement for this implementation. To achieve this in a grid-based environment, the dynamic constraints of the UAV need to be considered. With the constant speed assumption, the dynamic constraints are reduced to a minimum turning radius during flight. In order to achieve complete coverage, a UAV must then fly high enough such that the onboard camera FOV completely covers a cell while the UAV is executing a turn manoeuvre at this radius. This results in a lower limit for the constant search altitude.

A minimum ground sampling distance (GSD) to guarantee survivor detection was also imposed. This resulted in an upper limit on the search altitude, based on the camera resolution. This, along with the lower limit imposed by the UAV dynamic constraints, produced a range of feasible search altitudes. Therefore, the UAV and camera combination requires careful consideration.

A drawback of this method is that a significant amount of overlap is introduced, leading to considerable redundant coverage. The constant flying speed or constant altitude assumptions could be altered to address this. However, an advantage is that this overlap makes the coverage of the actual continuous environment more exact.

Fixed-wing UAVs were chosen for the project, since they have longer endurance, and SAR operations generally cover large environments. All the parameters associated with the UAVs and cameras in this project are assumed values based on literature, and would need to be confirmed using real-world experiments in order to be used in practice.

The divide areas algorithm for optimal multi-robot coverage path planning (DARP) was chosen to divide the grid-based search area into sub-regions for the UAVs. The division of the search area into sub-regions is what makes the approach a distributed approach. Based on the initial positions of the UAVs in the known environment, the algorithm iteratively assigns cells to those UAVs for searching. The algorithm seeks to optimise the area division by forming contiguous, equal-sized sub-regions.

One UAV initial position resides within each sub-region, and from this position, the UAV searches the associated region. A single robot coverage technique referred to as spanning tree coverage (STC) was used to achieve coverage of the individual regions. Dynamic constraints of the UAVs were also incorporated into the paths to make the paths feasible for real-world applications. The resulting closed-loop paths mean that once a UAV completes the coverage of its region, it is once again at its initial position.

A central deployment strategy was developed so that all the UAVs take off and land at the same location. This zone is referred to as the central ground station. The UAV initial positions were arranged in a perimeter configuration around this ground station and the area was assumed to be clear of obstacles for the manoeuvres in this space.

Four manoeuvres were described. Take off and landing are the manoeuvres to get from the ground to the search altitude, and vice versa. Departure and approach are the manoeuvres to and from the the UAV initial positions after take-off and before landing. The heading and dynamic constraints of the vehicle were considered for these manoeuvres. The closed-loop nature of the coverage paths mean that departure ends where approach begins, at the initial position of the UAV for the search.

In order to avoid collisions around the ground station, a flight schedule was created. The UAVs take off sequentially and land in the same order. Since the sub-regions are close to equal in size, the flight paths are of similar length, which eases this process. However, if necessary, an airborne wait cycle can be used so that one UAV can wait for another to finish landing before it begins its approach.

To take UAV endurance limitations into account, the sub-region sizes created by the divide areas algorithm were limited by UAV predicted flying time on one charge/refuel. A refuelling protocol was then developed that assigns more than one sub-region to a single UAV for searching. The UAVs would take off in sequence, complete searching respective sub-regions, and then land in sequence. They then refuel at the ground station and take off again to search the next unsearched sub-regions.

With the complete implementation developed, simulations were run to evaluate the effects of increasing the number of UAVs, the environment size, the obstacle density, and the number of refuels. Inputs such as the algorithm weights, the maximum iterations, the search altitude, the flying speed, and the type of UAV and camera used, were kept constant for these simulations.

The typical algorithm execution time and success rate was assessed. For the DARP algorithm, it was found that an increase in environment size or number of UAVs causes an increase in average algorithm execution time.

A slight disadvantage of the central deployment case is that this also increased the execution time on average for the same number of UAVs. The configurations were tested for up to eight UAVs, and in this case the increase was very slight.

The ability of the divide-areas algorithm to produce equal-sized sub-regions was evaluated by analysing the discrepancy value for different environment sizes, numbers of UAVs, obstacle densities, and deployment strategies. Obstacle density had the largest impact, causing more sub-optimal area divisions as the obstacle density increases. A slight increase in discrepancy was also found for the central deployment case when compared to the random UAV initial positions case.

When generating the coverage paths, it was found that spanning tree generation is the most time consuming portion of the process. In general, the execution time for coverage path generation increases with increasing environment size, but decreases with an increasing number of UAVs.

Algorithm failures were found to be directly related to the success of the DARP algorithm. A failure would occur in the event that there is no solution. There were, for example, more failures with higher obstacle densities. Obstacles may cause unreachable regions or generate environments where an even division of cells is impossible. A failure would also occur if the maximum iterations of the algorithm are reached before the solution is found. In this case, the algorithm weights or maximum iterations value could be altered to find the solution.

A disadvantage of the DARP algorithm is that it cannot distinguish between an environment with no solution and an environment where a solution exists, but the parameters need to be changed in order to find the solution. There is also no clear framework for selecting the algorithm weights and maximum iterations. Therefore in some scenarios it could be necessary to rerun the algorithm multiple times with different inputs. However, the overall time to find a solution is in the range of a few minutes, making it feasible for usage in the field, even if a few reruns are required. For an 8000 cell environment, representing a roughly 260 km^2 area, with eight UAVs and 10% obstacles, the DARP algorithm executes in under two minutes. The individual path generation algorithm for the same case takes less than 15 seconds. Even if the UAVs were decreased to two, the individual path generation takes under one minute to execute.

The survivor detection performance of the full system was tested and evaluated in simulation. Simulations were performed to determine the maximum time it would take to cover the search entire area using multiple UAVs. As expected, the maximum search time increased with environment size, and decreased with the number of UAVs.

For example, two UAVs searching 10000 cells (representing an area of roughly 320 km^2) would take roughly 35 hours to cover the entire search area, which could be impractical since they would have to search through the night. However, eight UAVs performing the same search would take less than 9 hours. These results were obtained for a Strix 400 UAV flying at a constant speed of 14 m/s. It would be up to the search coordination team to decide what combination best suites their needs, but more UAVs generally improve the chances of finding survivors sooner. The simulation results therefore showed maximum search times that are feasible for real-world SAR operations.

The path completion times with the Strix 400 UAV showed almost no change with the introduction of refuelling and central deployment. The perimeter configuration is therefore successful in keeping the departure and approach time relatively short, and preventing it from having a large impact on overall survivor detection times.

Four real-world environments were used as illustrative examples throughout the project. Two mountainous environments, a ground environment, and a marine environment were used. The Aberdeen example environment was used to show the statistical distribution of survivor detection times one could

expect for a ground SAR scenario. With one UAV it was found to take up to 17 hours. This time was halved as the UAVs are doubled. With eight UAVs the maximum survivor detection time was below 2 hours and 30 minutes. In search and rescue, most people that are found alive are located within the first 51 hours of going missing [80]. Therefore, these are reasonable survivor detection times in the context of a SAR operation.

The refuelling protocol was also evaluated in simulation by looking at the maximum energy consumption for searching a single sub-region. Energy consumption was represented by the time to complete a flight path, with safety factors applied to manoeuvres like turns to account for increased energy consumption with these motions. The Strix 400 UAV used for the simulations has a 9 hour flight time and the refuelling protocol had a near perfect success rate for keeping the energy consumption below this value.

As a whole the CPP strategy proposed to automate SAR with multiple UAVs was shown to be feasible. Simulations for theoretical and real-world examples show reasonable algorithm execution times, as well as favourable survivor detection times. Using more UAVs significantly improves survivor detection times and this could be invaluable for time sensitive SAR operations.

10.2 Recommendations for Future Work

This section lists possible improvements that can be made to the algorithms presented in this thesis, and ways that this research could be taken further. The following list briefly details potential future work:

- This implementation would benefit from a more rigorous investigation into environment mapping. Having a series of offline maps ready to use, or a system by which to map a new environment quickly would form a more complete solution for real-world application in SAR.

- The development of a methodology for assigning the connectivity and random component weights in the DARP algorithm is not currently available. An in-depth study on what weights are most appropriate for different environments would greatly simplify finding a solution.

- The assumption of a constant search altitude for this implementation limits the topographical variation that can be accommodated with this approach. An implementation that utilises a constant height above ground by following the topography would allow for a fairly constant GSD value, allowing more flexibility.

- The constant speed in this implementation assumption leads to large overlap values in order to accommodate dynamic constraints while also achieving complete coverage. Allowing for variations in speed would reduce redundant coverage. However, the advantage of this would have

to be weighed against the change in path completion time which directly impacts survivor detection time.

- Currently only static obstacles are accounted for. The addition of a short-term collision avoidance technique to address dynamic obstacles would make the implementation well suited for real-world application.

- Adapting the implementation for heterogeneous UAVs would make it easier for a search coordination team to utilize all their available resources. In the case of the divide areas algorithm, sub-region sizes would need to be adapted to represent the endurance limitations of different types of robots. Environment mapping in the event that they fly at different altitudes would also need to be adapted. This could even be taken further by incorporating unmanned ground vehicles (UGVs).

- Adapting the algorithms to accommodate a moving target would be useful in certain SAR scenarios, for example, if someone in the ocean was and being taken by a strong current. This may require the introduction of backtracking and possible re-planning. A more incremental planner may be necessary.

Appendices

Appendix A

Discretisation Tables

This appendix shows the full discretisation tables for the examples shown in Section 5.4. A summarised version for each scenario is shown in the section itself, but these show the full process in more detail. For a detailed description of the discretisation technique employed, see Section 5.2.

Important to note is that the flying height (H_f) represents height above sea level. The heights used in calculation of the square discretisation (l) would be relative to the highest topographic point in the environment ($h_{g_{\max}}$). Similarly, the height used to calculate the achieved GSD and overlap percentage would be relative to the lowest point in the environment ($h_{g_{\min}}$).

Each table has values describing the environment, mainly Δh_g, which is the actual topographic variation of the environment. Furthermore, there is the overall environment dimensions, represented by D_x and D_y.

The first table, Table A.1, summarises the discretisation of Spitskop Mountain in the Western Cape. It serves as an example of a low altitude mountainous terrain. The result of this discretisation, as well as the reasoning behind it, can be found in Section 5.4.1.

Similarly, Table A.2 summarises the discretisation of Champagne Castle in KwaZulu-Natal. This is an example of a high altitude mountainous terrain. Note how even though the overall topography of this application is steeper, the narrow region that is being searched actually has less resulting topographic variation than the low altitude example. The result of the discretisation can be found in Section 5.4.2.

Table A.3 looks at the area surrounding Aberdeen in the Northern Cape. It summarises the discretisation for a ground SAR scenario. Section 5.4.3 discusses this scenario and shows the final discretised environment.

Lastly, Table A.4 summarises the discretisation of a section of ocean off the coast of Jeffreys Bay in the Eastern Cape. It is an example of a marine SAR scenario and the resulting discretisation is detailed in Section 5.4.4.

Variable Name	**Value**	**Units**	**Equation**
GSD_{max}	4.7	cm/px	
ϕ_{max}	25.0	degrees	
H_{max}	366.6	m	5.5
$V_{\text{max}}(\Delta h_g = 0)$	21.7	m/s	5.12
V_f	14.0	m/s	
H_{min}	152.8	m	5.7
$(\Delta h_g)_{\text{max}}$	213.8	m	5.17
Δh_g	200.0	m	
$h_{g_{\text{min}}}$	300.0	m	
$h_{g_{\text{max}}}$	500.0	m	
$H_{f_{\text{min}}}$	652.8	m	5.9
$H_{f_{\text{max}}}$	666.6	m	5.9
H_f	660.0	m	5.9
FOV_x	164.1	m	5.2
FOV_y	109.7	m	5.3
r_{min}	42.8	m	3.7
l	89.8	m	5.14
D_x	15312	m	
D_y	7606	m	
GSD	4.6	cm/px	5.4
%Overlap	311	%	5.16

Table A.1: Table summarising the calculations and values necessary to discretise the Spitskop environment.

Variable Name	Value	Units	Equation
GSD_{max}	4.7	cm/px	
ϕ_{max}	25.0	degrees	
H_{max}	366.6	m	5.5
$V_{\text{max}}(\Delta h_g = 0)$	21.7	m/s	5.12
V_f	14.0	m/s	
H_{min}	152.8	m	5.7
$(\Delta h_g)_{\text{max}}$	213.8	m	5.17
Δh_g	177.0	m	
$h_{g_{\text{min}}}$	3200.0	m	
$h_{g_{\text{max}}}$	3377.0	m	
$H_{f_{\text{min}}}$	3529.8	m	5.9
$H_{f_{\text{max}}}$	3566.6	m	5.9
H_f	3535.0	m	5.9
FOV_x	162.1	m	5.2
FOV_y	108.3	m	5.3
r_{min}	42.8	m	3.7
l	88.6	m	5.14
D_x	11496	m	
D_y	5468	m	
GSD	4.3	cm/px	5.4
%Overlap	288	%	5.16

Table A.2: Table summarising the calculations and values necessary to discretise the Champagne Castle environment.

Variable Name	**Value**	**Units**	**Equation**
GSD_{max}	4.7	cm/px	
ϕ_{max}	25.0	degrees	
H_{max}	366.6	m	5.5
$V_{\text{max}}(\Delta h_g = 0)$	21.7	m/s	5.12
V_f	16.0	m/s	
H_{min}	199.5	m	5.7
$(\Delta h_g)_{\text{max}}$	167.1	m	5.17
Δh_g	144.0	m	
$h_{g_{\text{min}}}$	706.0	m	
$h_{g_{\text{max}}}$	850.0	m	
$H_{f_{\text{min}}}$	1049.5	m	5.9
$H_{f_{\text{max}}}$	1072.6	m	5.9
H_f	1060.0	m	5.9
FOV_x	215.4	m	5.2
FOV_y	144.0	m	5.3
r_{min}	56.0	m	3.7
l	117.8	m	5.14
D_x	15849	m	
D_y	7555	m	
GSD	4.5	cm/px	5.4
%Overlap	208	%	5.16

Table A.3: Table summarising the calculations and values necessary to discretise the Aberdeen environment.

Variable Name	**Value**	**Units**	**Equation**
GSD_{max}	4.7	cm/px	
ϕ_{max}	35.0	degrees	
H_{max}	366.6	m	5.5
$V_{\text{max}}(\Delta h_g = 0)$	21.7	m/s	5.12
V_f	19.0	m/s	
H_{min}	187.4	m	5.7
$(\Delta h_g)_{\text{max}}$	179.2	m	5.17
Δh_g	10.0	m	
$h_{g_{\text{min}}}$	0.0	m	
$h_{g_{\text{max}}}$	10.0	m	
$H_{f_{\text{min}}}$	197.4	m	5.9
$H_{f_{\text{max}}}$	366.6	m	5.9
H_f	215.0	m	5.9
FOV_x	210.3	m	5.2
FOV_y	140.6	m	5.3
r_{min}	52.6	m	3.7
l	115.0	m	5.14
D_x	3347	m	
D_y	1594	m	
GSD	2.8	cm/px	5.4
%Overlap	92	%	5.16

Table A.4: Table summarising the calculations and values necessary to discretise the Jeffreys Bay environment.

List of References

[1] "DJI And DroneSAR Bring Search And Rescue App To First Responders," nov 2016. [Online]. Available: https://www.dji.com/ie/newsroom/news/dji-and-dronesar-bring-search-and-rescue-app-to-first-responders

[2] H. Azpúrua, G. M. Freitas, G. Douglas, and M. F. M. Campos, "Multi-robot coverage path planning using hexagonal segmentation for geophysical surveys," *Robotica*, vol. 36, no. 2018, pp. 1144–1166, 2018.

[3] V. G. Nair and K. R. Guruprasad, "GM-VPC: An Algorithm for Multi-robot Coverage of Known Spaces Using Generalized Voronoi Partition," *Robotica*, vol. 38, no. 5, pp. 845–860, 2020.

[4] C. Rossi, L. Aldama, and A. ententos, "Simultaneous task subdivision and allocation for teams of heterogeneous robots," *Proceedings - IEEE International Conference on Robotics and Automation*, no. May, pp. 946–951, 2009.

[5] N. Hazon and G. A. Kaminka, "Redundancy, efficiency and robustness in multi-robot coverage," *Proceedings - IEEE International Conference on Robotics and Automation*, no. April, pp. 735–741, 2005.

[6] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos, "DARP: Divide Areas Algorithm for Optimal Multi-Robot Coverage Path Planning," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 86, no. 3-4, pp. 663–680, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10846-016-0461-x

[7] "Google maps." [Online]. Available: https://www.google.com/maps

[8] "Topographic maps." [Online]. Available: https://en-za.topographic-map.com/

[9] "Mapcarta." [Online]. Available: https://mapcarta.com/

[10] J. A. Guerrero and Y. Bestaoui, "UAV path planning for structure inspection in windy environments," *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2013.

[11] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss, "UAV-based crop and weed classification for smart farming," *Proceedings - IEEE International Conference on Robotics and Automation*, 2017.

[12] I. Maza, F. Caballero, J. Capitán, J. R. Martínez-De-Dios, and A. Ollero, "Experimental results in multi-UAV coordination for disaster management and civil security applications," *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2011.

[13] W. Chang, G. Yang, J. Yu, Z. Liang, L. Cheng, and C. Zhou, "Development of a power line inspection robot with hybrid operation modes," *IEEE International Conference on Intelligent Robots and Systems*, 2017.

[14] N. Basilico and S. Carpin, "Deploying teams of heterogeneous UAVs in cooperative two-level surveillance missions," *IEEE International Conference on Intelligent Robots and Systems*, 2015.

[15] H. X. Pham, H. M. La, D. Feil-Seifer, and M. Deans, "A distributed control framework for a team of unmanned aerial vehicles for dynamic wildfire tracking," *IEEE International Conference on Intelligent Robots and Systems*, 2017.

[16] L. Meiring and J. Engelbrecht, "Cooperative collision avoidance strategies for unmanned aerial vehicles," 2021. [Online]. Available: https://scholar.sun.ac.za

[17] H. Choset, "Coverage for robotics - A survey of recent results," *Annals of Mathematics and Artificial Intelligence*, 2001.

[18] "Iamsar manual volume i organization and management, international aeronautical and maritime search and rescue manual," 2016.

[19] "Iamsar manual volume ii mission co-ordination, international aeronautical and maritime search and rescue manual," 2016.

[20] "Virtual workshop on the establishment of an effective search and rescue (sar) organization."

[21] T. Leis, "The different types of search and rescue," 2021. [Online]. Available: https://sregear.com/blogs/news/the-different-types-of-search-and-rescue

[22] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen, *Search and Rescue Robotics*. Springer Berlin Heidelberg, 2008.

[23] R. Sisk, "Meet the us navy's robotic lifeguard named 'emily'," 2016. [Online]. Available: https://www.military.com/daily-news/2016/05/18/meet-the-us-navys-robotic-lifeguard-named-emily.html

[24] DroneSAR. (2019) DroneSAR NEW Features. [Online]. Available: https://fb.watch/8O5OMVjb45/

[25] V. S. Juan, M. Santos, and J. M. Andújar, "Intelligent UAV Map Generation and Discrete Path Planning for Search and Rescue Operations," *Hindawi*, vol. 2018, 2018. [Online]. Available: https://www.hindawi.com/journals/complexity/2018/6879419/

[26] S. Waharte, N. Trigoni, and S. J. Julier, "Coordinated search with a swarm of UAVs," *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, SECON Workshops 2009*, no. July, 2009.

[27] S. Waharte, A. Symington, and N. Trigoni, "Probabilistic search with agile UAVs," *Proceedings - IEEE International Conference on Robotics and Automation*, no. February 2014, pp. 2840–2845, 2010.

[28] A. Symington, S. Waharte, S. Julier, and N. Trigoni, "Probabilistic target detection by camera-equipped UAVs," *Proceedings - IEEE International Conference on Robotics and Automation*, no. February 2014, pp. 4076–4081, 2010.

[29] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," *Proceedings - EST 2010 - 2010 International Conference on Emerging Security Technologies, ROBOSEC 2010 - Robots and Security, LAB-RS 2010 - Learning and Adaptive Behavior in Robotic Systems*, no. June, pp. 142–147, 2010.

[30] P. Rudol and P. Doherty, "Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery," *IEEE Aerospace Conference Proceedings*, pp. 1–8, 2008.

[31] M. Wzorek, G. Conte, P. Rudol, S. Duranti, and P. Doherty, "From Motion Planning to Control - A Navigation Framework for an Autonomous Unmanned Aerial Vehicle," *The 21st Bristol UAV Systems Conference (UAVS)*, no. April, pp. 1–15, 2006. [Online]. Available: papers2://publication/uuid/45F9D4FA-ADE3-4D11-91EC-597F64909440

[32] S. Hayat, C. Bettstetter, and T. X. Brown, "Multi-objective drone path planning for search and rescue with quality-of-service requirements," 2020.

[33] S. Hayat, E. Yanmaz, T. X. Brown, and C. Bettstetter, "Multi-objective uav path planning for search and rescue," 2017.

[34] S. M. Lavalle, *Planning Algorithms*. Cambridge University Press, 2006.

[35] S. Russell and P. Norvig, *Artificial Intelligence: A modern approach*, 3rd ed. Pearson, 2016, vol. 48.

[36] H. Zhang, B. Xin, L. hua Dou, J. Chen, and K. Hirota, "A review of cooperative path planning of an unmanned aerial vehicle group," *Frontiers of Information Technology and Electronic Engineering*, vol. 21, no. 12, pp. 1671–1694, 2020.

[37] P. N. Atkar, A. Greenfield, D. C. Conner, H. Choset, and A. A. Rizzi, "Uniform coverage of automotive surface patches," *International Journal of Robotics Research*, 2005.

[38] N. Mir-Nasiri, J. Hudyjaya Siswoyo, and M. H. Ali, "Portable Autonomous Window Cleaning Robot," *Procedia Computer Science*, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.07.024

[39] E. M. Arkin, S. P. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry: Theory and Applications*, 1999.

[40] B. Englot and F. S. Hover, "Sampling-based coverage path planning for inspection of complex structures," *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pp. 29–37, 2012.

[41] I. A. Hameed, "Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain," *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2013.

[42] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, 2013.

[43] T. M. Cabreira, L. B. Brisolara, and R. Ferreira Paulo, "Survey on coverage path planning with unmanned aerial vehicles," *MDPI*, 2019.

[44] H. Choset and P. Pignon, "Coverage Path Planning: The Boustrophedon Cellular Decomposition," *Proceedings of International Conference on Field and Service Robotics*, pp. 203–209, 1997.

[45] H. Choset, E. Acar, A. A. Rizzi, and J. Luntz, "Exact cellular decompositions in terms of critical points of Morse functions," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 3, no. April, pp. 2270–2277, 2000.

[46] N. Nourani-Vatani, M. Bosse, J. Roberts, and M. Dunbabin, "Practical path planning and obstacle avoidance for autonomous mowing," no. January, 2006.

[47] B. Englot and F. Hover, "Planning complex inspection tasks using redundant roadmaps," *Proc. of the International Symposium of Robotics Research (ISRR)*, vol. 100, no. January, 2011.

[48] T. Danner and L. E. Kavraki, "Randomized planning for short inspection paths," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, no. April, pp. 971–976, 2000.

[49] A. Barrientos, J. Colorado, J. D. Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente, "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots," *Journal of Field Robotics*, vol. 28, no. 5, pp. 667–689, sep 2011.

[50] H. H. Viet, V. H. Dang, M. N. U. Laskar, and T. Chung, "BA: An online complete coverage algorithm for cleaning robots," *Applied Intelligence*, vol. 39, pp. 217–235, 2012.

[51] A. V. Le, V. Prabakaran, V. Sivanantham, and R. E. Mohan, "Modified a-star algorithm for efficient coverage path planning in tetris inspired self-reconfigurable robot with integrated laser sensor," *Sensors (Switzerland)*, vol. 18, no. 8, aug 2018.

[52] S. Dogru and L. Marques, "A*-Based Solution to the Coverage Path Planning Problem Robot 2017," *Advances in Intelligent Systems and Computing*, no. January, 2018.

[53] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Proceedings - IEEE International Conference on Robotics and Automation*, 1999.

[54] R. Nandakumar and N. Ramana Rao, "Fair partitions of polygons: An elementary introduction," *Proceedings of the Indian Academy of Sciences: Mathematical Sciences*, vol. 122, no. 3, pp. 459–467, 2012.

[55] C. Gao, Y. Kou, Z. Li, A. Xu, Y. Li, and Y. Chang, "Optimal Multirobot Coverage Path Planning: Ideal-Shaped Spanning Tree," *Mathematical Problems in Engineering*, vol. 2018, 2018.

[56] N. Baras, M. Dasygenis, and N. Ploskas, "Multi-Robot Coverage Path Planning in 3-Dimensional Environments," *2019 8th International Conference on Modern Circuits and Systems Technologies, MOCAST 2019*, pp. 0–3, 2019.

[57] X. Zheng, S. Jain, S. Koenig, and D. Kempe, "Multi-Robot Forest Coverage *," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[58] G. Even, N. Garg, J. Könemann, R. Ravi, and A. Sinha, "Min-max tree covers of graphs," *International Conference on Approximation Algorithms for Combinatorial Optimization*, 2003.

[59] G. S. C. Avellar, G. A. S. Pereira, L. C. A. Pimenta, and P. Iscold, "Multi-UAV Routing for Area Coverage and Remote Sensing with Minimum Time," no. November, 2015.

[60] C. Luo and S. X. Yang, "A real-time cooperative sweeping strategy for multiple cleaning robots," *IEEE International Symposium on Intelligent Control - Proceedings*, pp. 660–665, 2002.

[61] "How weather affects flight: Weather knowledge droneinfo," 3 2022. [Online]. Available: https://www.droneinfo.fi/en/study-material/how-weather-affects-flight-weather-knowledge

[62] Propeller Aero, "What is Ground Sample Distance (GSD) and How Does it Affect Your Drone Data ?" 2021. [Online]. Available: https://www.propelleraero.com/blog/ground-sample-distance-gsd-calculate-drone-data/

[63] R. C. Prim, "Shortest connection networks and some generalizations," 1957.

[64] N. Marwaha and E. Duffy, "Everything you need to know about digital elevation models (dems), digital surface models (dsms), and digital terrain models (dtms)," 2021. [Online]. Available: https://up42.com/blog/tech/everything-you-need-to-know-about-digital-elevation-models-dem-digital

[65] "U.s. releases enhanced shuttle land elavation data," 2022. [Online]. Available: https://www2.jpl.nasa.gov/srtm/

[66] H. W. Jurgens, I. Matzdorff, and J. Windberg, "International Anthropometric Data for Wok-Place and Machinary Design," vol. 108, pp. 8–9, 1998.

[67] "Wingtraone gen ii drone." [Online]. Available: https://wingtra.com/wp-content/uploads/Wingtra-Technical-Specifications.pdf

[68] "Warrior aero: Ocean seaplanes and gull uav specifications," 2021. [Online]. Available: https://www.warrioraero.com/gull-uav/specifications.html

[69] M. Speed, "Meet the albatross uav."

[70] "Sea cavalry sd 40." [Online]. Available: https://pdf.aeroexpo.online/pdf/xiamen-han-s-eagle-aviation-technology-co-ltd/sd-40/186008-8819.html#open35681

[71] "Meet the avem." [Online]. Available: https://www.aeromapper.com/avem-2-2/

[72] "D-sentry up caeli via." [Online]. Available: https://www.up-caelivia.it/d-sentry

[73] "Strix400." [Online]. Available: https://pdf.aeroexpo.online/pdf/eos-technologie/strix-400/187591-21258.html#open66821

[74] "Flir vue pro." [Online]. Available: https://www.flir.com/products/vue-pro/

[75] "Mavic air 2 - specifications," 2022. [Online]. Available: https://www.dji.com/mavic-air-2/specs

[76] W. Kocay and D. L. Kreher, *Graphs, Algorithms and Optimization*. CRC Press, 2004,.

[77] H. Huang, A. V. Savkin, and W. Ni, "Energy-efficient 3d navigation of a solar-powered uav for secure communication in the presence of eavesdroppers and no-fly zones," *Energies*, vol. 13, 2020.

[78] Y. Li, H. Chen, M. Joo Er, and X. Wang, "Coverage path planning for UAVs based on enhanced exact cellular decomposition method," *Mechatronics*, vol. 21, no. 5, pp. 876–885, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.mechatronics.2010.10.009

[79] G. W. Milne, *Navigation Work Package*. University of Stellenbosch, 2003.

[80] "Ohsu researchers find time is best predictor of survival in search and rescue missions," 2007. [Online]. Available: https://news.ohsu.edu/2007/07/17/ohsu-researchers-find-time-is-best-predictor-of-survival-in-search-and-rescue-missions