POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria
Master of Science in Computer Science and Engineering

A DIVIDE-ET-IMPERA APPROACH TO PATH PLANNING FOR GROUND COVERING WITH AN UAV

Supervisor:
Prof. Francesco AMIGONI

Master Thesis by:
Simone BIANCHI
Student ID 883360

Academic Year 2019-2020

Abstract

The Coverage Path Planning (CPP) problem is the problem of determining a path that a robot, in our case an UAV (Unmanned Aerial Vehicle), must follow in order to cover with its sensors all the points of a target area. In general, 3D CPP problem is a CPP problem in a 3D environment. The application scenarios of the 3D CPP are several: surveying, precision agriculture, structural inspections, covering of ocean floors.CPP problems are known to be NP-hard. Due to this, optimal solutions are approximated using various methods.In particular, the problem addressed in this thesis is to cover a 2D area at the ground by solving multiple CPPs over 2D grids, that are obtained as the result of the intersection of the 3D environment with $|H|$ horizontal planes at different heights $h \in H$, where H denotes the set of discrete heights. The UAV uses a sensor with a conic FOV (Field Of View) that is used to cover the ground.
Our main goal is to reduce the computational cost in a CPP problem. To do this, we adopt a divide-et-impera approach. In computer science, divide-et-impera is a method that consists in dividing the initial problem into two or more simple sub-problems. After that, the solutions of the sub-problems must be combined to obtain the final result. In our case, we divide the entire environment in smaller sections that are covered separately through the Art Gallery Problem (AGP) as a set cover problem in order to find a feasible set of covering points (from which the target area can be fully covered) and the Travel Salesman Problem (TSP) to connect them. After the paths over all the zones are produced, a merge algorithm obtains the final single path that covers the entire target surface.
We introduce two different merge algorithms. The first one solves the problem more rapidly in terms of computational time but generally produces high cost paths, vice versa the second one produces paths with less cost than the first algorithm at the expense of a higher computational time. We implement these algorithms in three different environments and with different FOVs (Field Of Views) of the UAV sensor. Then we compare the results obtained with the results obtained without a divide-et-impera approach.

Sommario

Il problema del Coverage Path Planning (CPP) consiste nel determinare un percorso che un robot, nel nostro caso un UAV (Unmanned Aerial Vehicle), deve seguire per coprire con i suoi sensori tutti i punti di interesse evitando ostacoli. Il 3D CPP è la versione del problema del CPP in ambienti 3D. Il 3D CPP è fondamentale in molte applicazioni, come la sorveglianza, la ricostruzione di strutture 3D, e il controllo delle semine in agricoltura. Il problema del CPP è NP-hard, per questo motivo vengono proposte soluzioni che approssimano il risultato ottimo. In questa tesi trattiamo il problema di copertura di un'area al suolo come una composizione di più problemi di copertura 2D, poiché suddividiamo l'ambiente 3D in $|H|$ sezioni orizzontali a diverse altezze prestabilite $h \in H$, dove H è l'insieme discreto delle altezze. L'UAV utilizza un sensore con un FOV (Field Of Viewe) conico usato per coprire il terreno.
Il nostro obiettivo principale è quello di ridurre il costo computazionale nei problemi di CPP. Per fare questo utiliazziamo un approccio divide-et-impera. In informatica, tale approccio prevede la suddivisione del problema iniziale in uno o più sottoproblemi più semplici. Successivamente, le soluzioni dei sottoproblemi devono essere combinate a formare la soluzione del problema iniziale. Nel nostro caso, dividiamo l'intera mappa in diverse zone più piccole che vengono coperte separatamente. Nel nostro caso, la copertura viene effettuata da un Art Gallery Problem (AGP), un problema di set cover il cui obiettivo è quello di trovare un insieme di punti di copertura che coprano l'intera area di interesse. Successivamente, il Problema del Commesso Viaggiatore (TSP) viene utilizzato per trovare un tour che connetta i punti di copertura individuati. Dopo aver ottenuto i percorsi di copertura di tutte le zone, un algoritmo di unione produce il risultato finale.
Due algoritmi di unione vengono proposti nella tesi. Il primo produce risultati con un minore tempo computazionale ma generalmente produce percorsi costosi per quanto riguarda la distanza, viceversa, il secondo algoritmo ha con un costo computazionale più elevato ma minori costi di distanza. Implementiamo questi due algoritmi di unione su diversi ambienti e con differenti FOV (Field Of View) del sensore dell'UAV. Infine analizziamo e compariamo i risultati ottenuti mettendoli a confronto con i risultati ottenuti senza un approccio divide-et-impera.

Ringraziamenti

Desidero ringraziare il Professor Francesco Amigoni per la disponibilità mostratami e per gli utili consigli datomi durante questi ultimi mesi.

Ringrazio la mia famiglia, in particolare i miei genitori Beatrice e Marco per avermi sostenuto economicamente durante questi anni sia economicamente che moralmente nei momenti più stressanti e in quelli più gioiosi. Un ringraziamento particolare va a mio papà Paolo che mi ha trasmesso la passione per la tecnologia sin da bambina e senza di cui non avrei mai intrapreso questa strada. Ringrazio anche mio zio Flavio che mi ha sempre accompagnata, letteralmente, in questi anni.

Ringrazio Laura per esserci sempre stata nei momenti belli e in quelli più tristi, per avermi sempre rallegrato dopo un insuccesso e per avermi sempre spronato ad affrontare qualsiasi difficoltà con sicurezza e a credere nelle mie capacità.

Infine, ringrazio i miei colleghi universitari che hanno reso più divertente e piacevole questo percorso.

Contents

List of Tables

List of Figures

Acronyms

Unmanned Aerial Vehicle (UAV)
Coverage Path Planning (CPP)
Spanning Tree Coverage (STC)
Autonomous Underwater Vehicles (AUV)
Art Gallery Problem (AGP)
Travelling Salesman Problem (TSP)
Neighborhoods-TSP (TSPN)
Field of view (FOV)
Coverage sampling problem (CSP)
Probabilistic Roadmaps (PRM)
Rapidly-exploring Random Tree (RRT)

Chapter 1

Introduction

The applications of UAVs (Unmanned Aerial Vehicles) have increased over the last years. While they originated mostly in military applications, their use has rapidly expanded to leisure activities, such as photography and videography, applications in industrial inspection [6], in agriculture (in precision agriculture for crop, soil, and irrigation monitoring) [4, 40] and for the inspection of buildings. Also, another type of robots, called AUV (Autonomous Underwater Vehicle) are used to cover seafloors [1,19]. The advantages of UAVs and AUVs are: lower risks for humans and more efficiency and precision in performing the tasks.

In the field of autonomous robot planning, the Coverage Path Planning (CPP) problem is a well studied problem. It consists of finding a path that covers a target area avoiding obstacles. The space may be represented in different ways [1,10,42,44] and the paths can be computed using different methods [18,35,43,44]. Aiming to cover 3D environments, a number of algorithms have been recently proposed [20,26]. Some methods are based on a two-step optimization process and are independent from any specific scenario. In the first phase, an AGP (Art Gallery Problem) is solved, which consists of finding the minimal set of viewpoints (called covering points) from which the whole target space can be covered. In the second phase a Travelling Salesman Problem (TSP) is solved, and the shortest tour connecting the covering points is computed. [3, 39] propose a two-phase optimization methods using MILP (Mixed Integer Linear Programming) for solving a CPP in 2D environments. Due to the fact that both the AGP and the TSP are NP-hard, fast algorithms that provide approximate solutions are used in many works to solve these two problems: for istance Latombe and Gonzalez-Banos in [22, 23] propose two approximation solutions to solve the AGP as a set cover problem.

In this thesis, a divide-et-impera approach is studied in order to improve the computational cost solution of CPP problems. In particularly, our goal is to cover

a 2D terrain (like an agricultural field) with an UAV that moves in a 3D space. To cover an area, the UAV has a sensor with a conic shape projecting to the ground a circle of radius $r = h \cdot \tan(\alpha)$ where α is half of the apex angle. Given a discrete set of heights, we decompose the 3D environment in different horizontal cross sections, represented as grids, and we solve multiple 2D CPPs. In our case the paths start from a fixed point at the ground (called depot). Our approach divides the initial problem into different sub-problems. We divide the environment in zones with different characteristics. The classification of the types of zone was introduced by Amigoni and Riva in [2] and by Ghiotti in [21]. They classified environments into two types:

- Type 1 comprises the environments in which an UAV covers a larger area when going up, this means that the area covered at given height is always included in the area covered at a higher height, as in an open field.
- Type 2 includes the environments in which the area that can be covered at given height is a subset of the area covered at any lower heights, as for an industrial area with sheds.

Type 1 zones are covered by a SingleHeight algorithm that produces paths over a fixed height amd selects the one with the minimum cost. Type 2 zones are covered by a TwoHeight algorithm that computes tours over two heights with a hierarchical approach, starting from the highest grid and going down, to cover incrementally the area left uncovered by tours at upper heights. Then it compares all the tours and selects the minimum-cost tour.

Every zone is covered separately as an independent environment. Firstly, a set of covering points is founded by the AGP using a greedy approach, then a TSP connects together the points, using Theta* to calculate the distance cost.

When all the zones are covered we merge the paths to produce a single path over the entire environment. A similar problem is faced by Galceran et al. in [19] where they used a divide-et-impera approach to cover the seafloor with an AUV (Autonomous Underwater Vehicle). We propose two different merge algorithms.

Merge Algorithm 1 selects one path as first and the other path as second. The edge that merges the two paths together links the last point of the first path to the first point of the second path.

Merge Algorithm 2 selects one path as *external* and the other as *internal*. All the points of the two paths are analyzed in a greedy way and the solution with the minimum distance cost will be selected. Calling (p_1, p_2) the points selected for the merge, we will have a final path that goes through the *external* path until

p_1 is reached. Then goes to p_2 and goes through all the *internal* path and returns to the *external*, returning to the depot.

We implement both algorithms on three different environments with different UAV sensor's FOVs. We compare the results obtained with our divide-et-impera approach with the results obtained with a single CPP over the entire target area.

Merge Algorithm 1 produces results in less computational cost but with a high distance cost. On the other hand, Merge Algorithm 2 has an high computational cost but produces better results in terms of distance.

1.1 Structure of the Thesis

The thesis is structured as follows:

- Chapter 2 discusses the state of art of CPP. More specifically: firstly, it introduces various coverage methods for 2D environments and then works about 3D CPP and multi-robot CPP are discussed.
- Chapter 3 introduces the problem studied in this thesis. We start describing the problem statement, how the environment is represented and the characteristics of the UAV. We analyse different types of environments and how a solution is obtained with our divide-et-impera approach.
- Chapter 4 outlines the divide-et-impera process in all its steps. How the division is done, the AGP algorithm and the TSP algorithm used. Then two merge algorithms are described.
- Chapter 5 introduces the simulation environment and the experimental tools. Then we describe and comment the results obtained.
- Chapter 6 summarizes the obtained results and proposes some possible future improvements.
- Appendix A illustrates occupancy grid maps of the environments used to test the algorithms.
- Appendix B illustrates the maps with the optimal path using both the merge algorithm for every environment and FOV.

Chapter 2

State of the Art

This chapter discusses the most relevant works concerning Coverage Path Planning (CPP) and will try to compare them to this study. More precisely, the survey of the works is divided in 2D and 3D approaches.

2.1 Introduction to CPP

Coverage Path Planning (CPP) is the task of determining a path along a set of points to cover an area or a volume of interest while avoiding obstacles.
There is a large variety of CPP algorithms and many different classifications of these. The most important classification is the one that divides the algorithms between the ones that operate a 2D coverage and the others that offer a 3D one. In the middle of these two groups are those algorithms that are used to cover a 2D surface while moving in a 3D space. In this thesis we refer to this particular class of CPPs.
Another possible classification is between online and offline algorithms. The former, take real-time measurements, decisions are calculated and used during the execution of the algorithm, the latter, have a complete knowledge of the environment a priori, before execution.
Algorithms can also be divided basing on the type of the trajectory between points, fixed a priori or computed. For example, fixed trajectory consider a fixed track like a spiral or a circle, while computed trajectory is demanded to an algorithm. Often, the fixed trajectory algorithms are also offline but this is not always the case (our approach is offline with a computed trajectory). Another classification is on the type of coverage they offer (uniform or non-uniform). In non-uniform coverage some areas of the target area are skipped because not relevant, while this do not happen in uniform coverage. CPP is done by a single robot or by two ore more, in this case we talk about a swarm of robots that interact between them to reach a common goal.

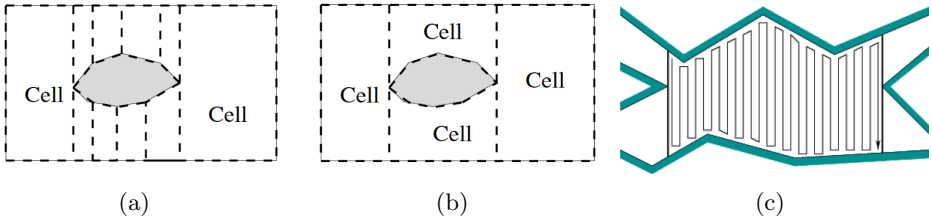


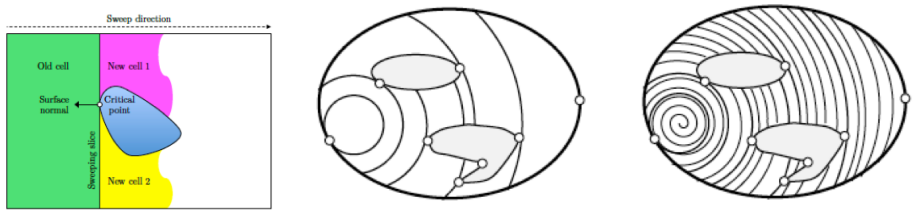
Figure 2.1: (a)Example of trapezoidal decomposition [10]. (b) The boustrophedon decomposition [10]. (c) A path using boustrophedon decomposition [10].

Environments can be modelled in different ways: polygonal as [10], grid-discretized as in [44], graph-based as in [42]. In particular, 2D model can be planar surface in the case of a floor cleaning, lawn moving, land mine detection, etc. In real scenarios, CPP is often done over 3D environments, for example in the case of UAVs covering fields, AUVs (Autonomous Underwater Vehicle) covering seabeds, or robots which are used to inspect the surface of objects like caves or buildings. Sometimes 3D environments can be simplified using 2D planar surfaces at different heights like in [26] and in our thesis.

2.2 2D Coverage

Concerning 2D scenarios, Choset and Pignon [10] proposed an off-line, fixed trajectory approach: the boustrophedon cellular decomposition. It is an extension of the simplest cellular decomposition: the trapezoidal approach, in which the free space in the environment is divided in cells of trapezoidal shape. A drawback of the trapezoidal approach is that it generates many cells that can be merged to form bigger cells as shown in Figure 2.1(a). The boustrophedon cellular decomposition resolves this problem. Indeed, the free space is decomposed in non-overlapping regions, called cells, formed extending a vertical line both up and down from a vertex and choosing only the ones in which the line does not touch an obstacle in both the directions as shown in Figure 2.1(b). Then an adjacency graph between the cells is computed and, in each cell, a coverage path is found by means of back-and-forth motions. Finally, a path from one cell to another through the adjacency graph is reckoned as shown in Figure 2.1(c). This approach works only with polygonal obstacles.

In [1] the authors starting from the boustrophedon cellular decomposition proposed an evolution based on critical points of a Morse function, i.e., a Morse function is one that has non-degenerate critical points. A critical point is a point



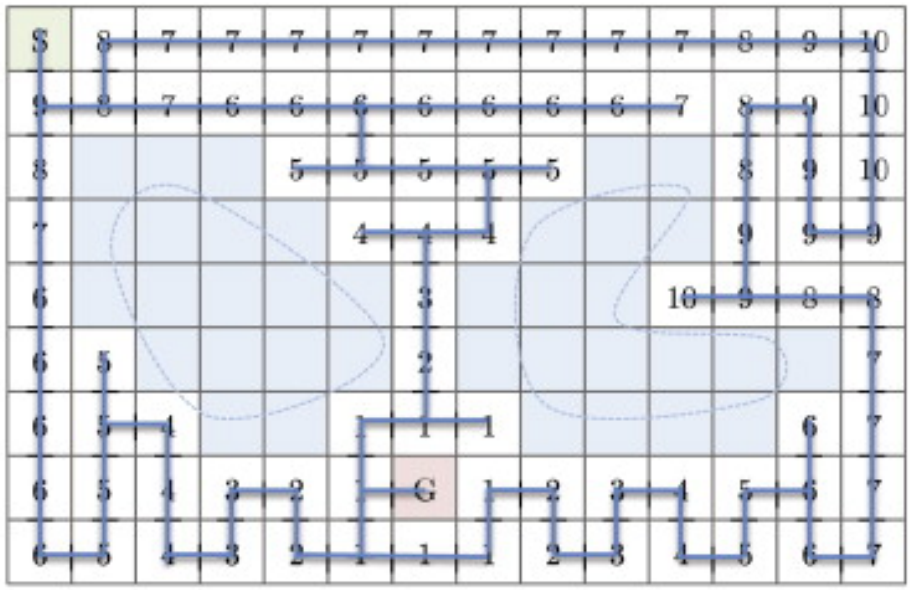
(a) (b) (c)

Figure 2.2: (a) Cell determination using the Morse based decomposition [1]. (b) and (c) Morse cellular decomposition for $h(x) = x_1^2 + x_2^2$ in which the slices are circles. Rather than moving along circular paths and stepping outward, the robot follows a spiral pattern [1].

$\rho \in \Re^m$ and exists a function $h : \Re^m \to \Re$ that has all its partial derivatives are 0. The point ρ is non-degenerate if and only if its Hessian is not singular. Considering a single variable function, a critical point corresponds to local maximum, local minimum, or an inflection, while non-degenerate critical points mean points that are a maximum in some directions and a minimum in others. The authors defined a slice as a "pre-image of a real-valued function", taking cue from Canny [8] method in which a slice is a codimension one manifold. While boustrophedon method extends cell lines from the vertices of the obstacles, the Morse decomposition traces the cell boundaries where a connectivity changes of the slice occurs in the free space. This fact occurs at critical points of the Morse function in correspondence of obstacle boundaries. Different cell shapes and different coverage path patterns can be obtained choosing different Morse functions, as shown in Figure 2.2(b) and Figure 2.2(c). After the cell decomposition phase, a path in the adjacency graph is computed and a coverage path in each cell is generated.

While these two methods divide the environment in cells of different shapes, there are also methods that rely on a grid based decomposition, where each cell is equal to another one. This is also the method that we use for our thesis.

Zelinsky [44] presented an off-line grid based coverage method extending the distance transform path planning method proposed by [27], used to find a path from a start to a goal. The planner propagates a wavefront from the goal cell to the starting cell through all free space and around obstacles. A value is assigned to each cell, starting from the goal with 0 and giving 1 to all the neighbours (a neighbour cell is a cell directly linked, in the eight directions, to the one considered) of the goal, and 2 to all the neighbours of the cells marked with 1 and so



(a)

Figure 2.3: An example of wavefront path [44].

on until the initial cell is reached. Then, from the starting cell a coverage path can be found choosing every time the highest unvisited neighbouring cell until the goal is reached as shown in Figure 2.3(a).

Gabriely and Ramon [18] proposed the Spiral STC (Spanning Tree Coverage) algorithm, an online approach with a fixed shape of the path as a spiral. The algorithm incrementally subdivides the environment in a grid of 2D-size cells, each one divided in four smaller cells of D-size as represented in Figure 2.4(a). Starting from the current cell, the robot selects the new 2D-cell to visit in anti-clockwise direction and adds it to the spanning tree as new edge. This process is repeated until no new neighbours are discovered. This coincides with the fact that the robot has reached the end of one side of the spanning tree. At this point it turns around and traverses the other side of the tree returning to the starting point. An example of the path computed by Spiral STC algorithm is shown in Figure 2.4(b).

Another approach based on 2D grid representation is the one proposed by Riva and Amigoni in [35]. They propose a CPP based on a Greedy Randomized Adaptive Search Procedure (GRASP). The GRASP was first introduced by Feo

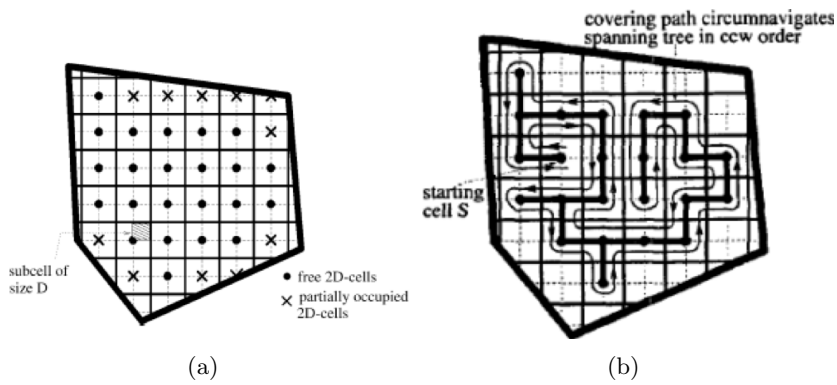


Figure 2.4: (a) (b) A solution using Spiral STC algorithm [18].

and Resende in [17], and it is a methauristic approach in which each iteration is composed of two phases: construction and local search. The construction builds a feasible solution, and then the local search tries to improve it. The GRASP method for CPP proposed by Amigoni and Riva first calculates an initial coverage tour S_{init} based on a greedy approach. The greedy approach, at each iteration, randomly selects a vertex from the set of best vertices that can be reached from the current node, ordered according to a combination of distance and expected covered area. The local search procedure consists of two consecutive phases: exchange and removal. During an exchange a vertex in S_{init} is substituted with a vertex not in the tour. At the end, when all the possible substitutions have been calculated, the substitution S_{l1} that produces the maximum decreases of the tour cost is selected. Then, removal excludes one vertex at once from S_{l1}, and if the solution is still feasible computes the new tour. Again, among all the new feasible solutions found by removal, the one with the maximum decreases of tour cost is selected. In such a way a local optimum is reached.

Tomoioka et al. in [39] propose an offline grid-based CPP method for mobile surveillance. Their method is based on MILP (Mixed Integer Linear Programming). First they decompose the target area in a grid, then a graph that considers the possible routes, on the basis of the camera orientation, is constructed. The tour is computed over it solving a MILP problem with the objective of minimizing the total travel time of the tour. To guarantee the optimum and find a feasible tour some constraints must be formulated:

- any target cell is observed from at least one camera candidate,
- the incoming flow of each vertex is equal to the outgoing flow to guarantee closed routes. (The flow $f(e)$ is a non-negative integer variable that

represents the number of times the generated route passes through edge e),

- travelling route constraints for the TSP formulation.

To reduce the solution space without losing any optimal solution, the authors propose to cluster groups of free cells in large problem instances.

Recently, advanced algorithms based on a two-step optimization process have been presented. In the first phase an AGP (Art Gallery Problem) is solved, which corresponds to find the minimal set of viewpoints (called guards or covering points) that cover the whole free space (see Lee and LIn in [28]). In the second phase Travelling Salesman Problem (TSP), firstly introduced by Dantzig, Fulkerson, and Johnson in [14], is solved, which consists in computing the shortest tour to connect the points selected by the AGP.

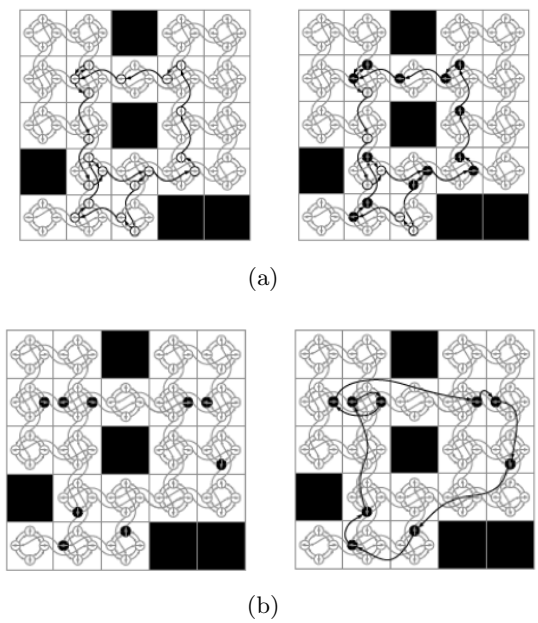


Figure 2.5: (a) Optimal tour when travelling time is minimized first [3]. (b) Optimal tour when sensing time is minimized first [3].

Arain et al. [3] propose different offline grid-based algorithms to solve a problem of sensor placement for gas detection. The authors limit the movement of the robot whit a finite set of poses $p_i = (\alpha_i, \theta_i)$, where α_i is a free cell of the space and θ_i is an allowed orientation. To compute the shortest path, the authors build a movement graph, to represent the possible movements of the robot in each cell

of the grid. They define a candidate sensing configuration c_i as a tuple (p_i, ϕ_i, r_i), where p_i is the robot pose, ϕ_i is the central angle of a circular sector (this circular sector represents the view of the robot), and r_i its radius. To define the cells observable in the circular sector with center in p a visibility function v_p is defined. They formulate different approaches and compare them. First, they define the problem as an optimization problem by formulating it as a MILP (Mixed Integer Linear Programming) problem inspired by the work of Tomioka et al. [39]. They formulate it as a combination of a Watchman Route Problem (WRP) and an AGP. But, this formulation becomes practically unfeasible as the number of covering points grows, because both the WRP and AGP are NP-hard.
So, they consider two variations of a two-step approach:

- In one case, first selects the tour, with the minimum travelling time, that cover all the target area. And then picks the minimum set of covering points among the points of the chosen tour. This process is shown in Figure 2.5(a).
- The other approach first finds the minimum set of covering points, and then uses the TSP to compute the shortest tour as in Figure 2.5(b).

In both cases the problem has a bottleneck in finding the set of minimum covering points solving the following integer linear programming problem:

$$\begin{aligned} \text{minimize} \quad & C^T T_s \\ \text{subject to} \quad & VC \geq 1 \\ & C \in \{0, 1\} \end{aligned}$$

Let C_P be the set of candidate sensing configuration, T_s is a column vector of size $|C_P|$ that represents the sensing cost associated to each covering points, V is a binary matrix of size $n \times |C_P|$, where n is the number of cells in the problem. $V[\alpha, c]$ is equal to 1 if $\alpha \in v_p(c)$, 0 otherwise. And C is a column vector of cardinality $|C_P|$, whose elements are binary variables representing if a given candidate sensing configuration is selected or not. The main contribution of their work is to propose the conv-SPP algorithm, a variation of the previous integer linear programming problem to quickly finding the minimum set of covering points. It is based on a convex relaxation which introduces sparsity, so it drastically reduces the number of variables. Furthermore, it generates results very close to the optimal ones.

Table 2.1 summarizes all the methods presented, with the most important characteristics in the foreground.

Reference	Algorithm	Offline/Online	Trajectory
[10]	Boustrophedon cellular decomposition Adiacency graph	Offline	Fixed (back-and-forth motion)
[1]	Morse-based decomposition Adiacency graph	Offline	Computed (depends on the cell shape)
[44]	Grid based decomposition Wavefront	Offline	Computed
[18]	Grid based decomposition Spanning Tree Coverage	Online	Fixed (spiral)
[35]	Grid based decomposition GRASP (greedy + local search)	Offline	Computed
[39]	Grid based decomposition MILP solution	Offline	Computed
[3]	Grid based decomposition AGP + TSP	Offline	Computed

Table 2.1: Summary of 2D works presented.

2.3 3D Coverage

In the last years more and more UAVs are used in the CPP subject. In these cases the environment is no more a 2D one, but it's a 3D dimension space, in which the robots can move not only in a plane but they can move in a space. In these environments 3D CPP is required, therefore the previous methods cannot be applied straightforwardly. In the field of 3D CPP the concept of 3D structural inspection is relevant in which it is necessary to cover 3D-surfaces as boundaries of buildings, ocean floors, agricultural fields, or automotive parts.

In [26] Hert et al. used a 2D planar algorithm to solve 3D CPP. They propose an on-line approach for an AUV to cover an unknown underwater environment. This approach consists in applying a 2D planar algorithm in the successive horizontal planes laying at different depths. In essence this means that the 3D surface is divided in successive planes at different depths and the intersection points between the planes and the 3D surface that must be covered, are projected onto the 2D plane. Robot adjusts its height so as to maintain a constant distance from the ocean floor. A constraint on the environment is formulated as any vertical line passing through the surface intersects it at exactly one point. Consequently there is a one-one correspondence between the points in surface and those in the x-y

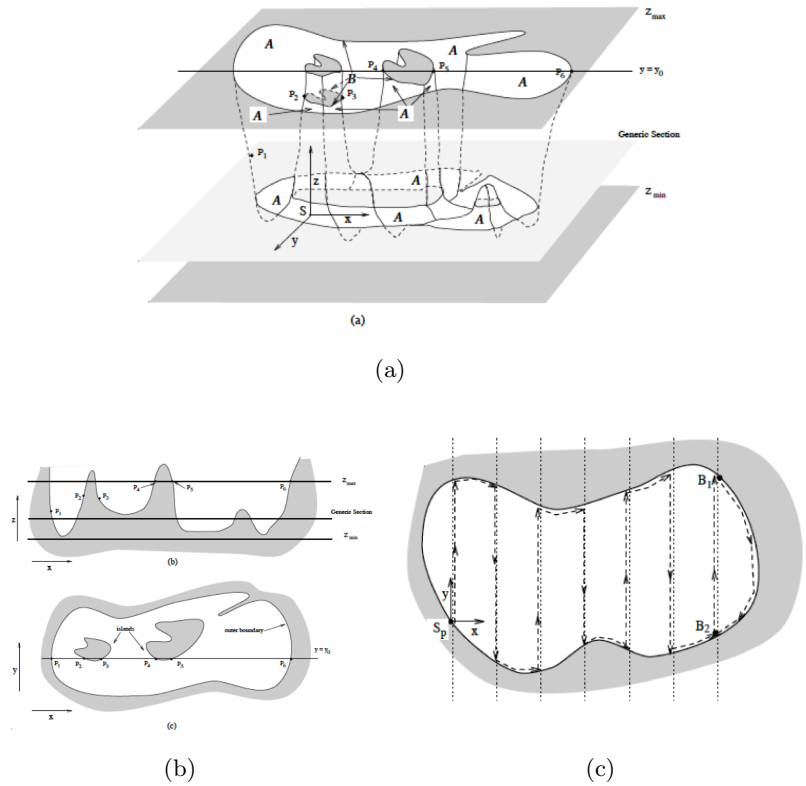
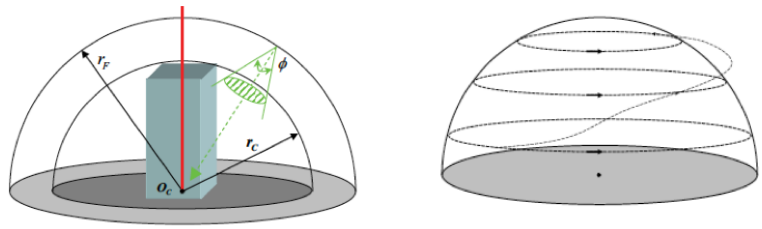


Figure 2.6: (a) The robot will cover area A with boundary B [26]. (b) At the top is shown a cross section in the plane $y = y_0$, P_1, P_2 and P_3 are points where the surface exceed the threshold slope μ. In the bottom the area A is projected onto the 2D plane [26]. (c) Shows the path of the robot starting from S_P [26].

plane and some elements as canvas cannot be present in this type of environment. This kind of non planar surface are called *vertically projectively planar surfaces*. This process is shown in Figure 2.1(a) and Figure 2.1(b). The considered 3D surface is bounded between two threshold surfaces, $z = z_{min}$ and $z = z_{max}$. Each plane is represented using a semi-approximate cellular decomposition, in which the space is divided in vertical slices of the same width. The planar environment is covered zigzagging along parallel straight lines and starting from any point in the space as shown in Figure 2.6(c). In the projectively planar 3D environment the same zigzagging motion leads the robot from one grid plane to the next. The only addition is a vertical movement to pass from one grid to another to take into account the changes in the height of the terrain.
In our algorithm the 3D space, is divided in successive planes, but the planes are represented as equivalent grids at different heights with different occupancy



(a) (b)

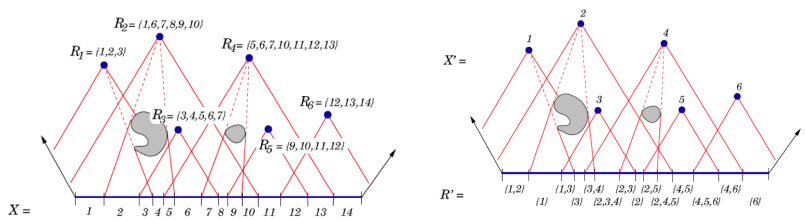
Figure 2.7: Hemispherical simplification of urban structure and hemispherical trajectory of UAV [9]

values. This removes the restriction of a vertically projectively planar surface and does not impose a zigzagging behavior.

Another approach with fixed trajectory is proposed by Cheng et al. [9]. The authors propose an offline coverage algorithm for urban structures, the buildings with geometric solids such as cylinders and hemispheres. This simplification allows the authors to consider the problem as a problem of covering regular non-planar surfaces, as shown in Figure 2.7(a). In the hemispherical model the structures are approximated with a coverage hemisphere $H_C(O_C, r_C)$. Also the UAV trajectory is fixed, since it has to follow a flight hemisphere $H_F(O_C, r_F)$ where the center O_C coincides with the center of the coverage hemisphere, while it holds that $r_F > r_C$ meaning that the flight hemisphere is bigger than the coverage one. Intuitively, the UAV is moving along a sequence of horizontal circles at different altitudes to cover the area as shown in Figure 2.7(b). Similarly to our work they consider an UAV with conical FOV (Field Of View).

Bircher et al. [6] propose a two-step offline optimization algorithm to solve a structural inspection problem, a sub-problem of CPP which consists in finding a path that covers the whole surface of a desired structure. In this paper the 3D structure is represented by triangular meshes. The first step consists in selecting the viewpoints without following an optimization strategy, but trying to make the connecting path as short as possible. The idea behind this is that sometimes, due to a continuous sensing device, it is more important the position in the space of the viewpoints than their number. So, the authors only select a feasible view point for each triangle.

In many coverage problems due to the NP-complexity of the AGP, a variation



(a) (b)

Figure 2.8: Sampling scheme and its dual scheme proposed by Latombe and Gonzalez-Banos to solve a variation of the AGP [23].

of the AGP based on random sampling and introduced the first time by Latombe and Gonzalez-Banos in [22] and [23] has been used. Latombe and Gonzalez-Banos propose two approximations to solve the AGP as set covering problems for sensor placement to reconstruct a 3D image. They solve this problem assuming that a 2D layout of an horizontal cross-section of the environment is given.
The first algorithm samples the workspace W at random to have a set of guards candidates, G, and selects the subset with minimum cardinality. Then a greedy algorithm is applied to choose at each step the guard with the highest coverage of the uncovered boundary. First, let X be the set of the elements that must be covered, $R = \{R_1, R_2, ..., R_M\}$, where R_i is the set of elements of X covered by element $g_i \in G$, this is shown in Figure 2.8(a). The set R_i with highest cardinality is selected and removed from R and the elements of R_i are removed from X and from the other $R_j \in R$, with $j \neq i$. The process is repeated until X is empty.
The second algorithm is based on a dual sampling scheme, shown in Figure 2.8(b). In the basic version of the random sampling algorithm, the complexity grows w.r.t. the number of samples. Instead, in the dual algorithm, the elements that must be covered are sampled, to have the possibility to vary the number of guards. So this time a point is selected from the unseen portion of the perimeter and the region from which such a point is visible, $O(p)$, is computed. Then $O(p)$ is sampled to find the position with highest coverage and a new guard is selected. The process is repeated until the unseen boundary is small enough.

The AGP based on random sampling and reduced to the set cover problem explained before, is called *Coverage Sampling Problem* (CSP) by Englot and Hover [16]. They define the CSP as the problem of finding a feasible covering set. The *watchman route algorithm using dual sampling* proposed by Danner and Kavraki [13] and the *reduntant roadmap algorithm* proposed by Englot an Hover in [15] are two examples of CSP. They use a range space representation of the en-

vironment, that consists in a set system (P, Q), where P is a finite set of geometric primitives of the structure that must be covered, and Q is the robot configuration space. These two algorithms and their stateflow diagram are illustrated in Figure 2.9, which is taken from [16].

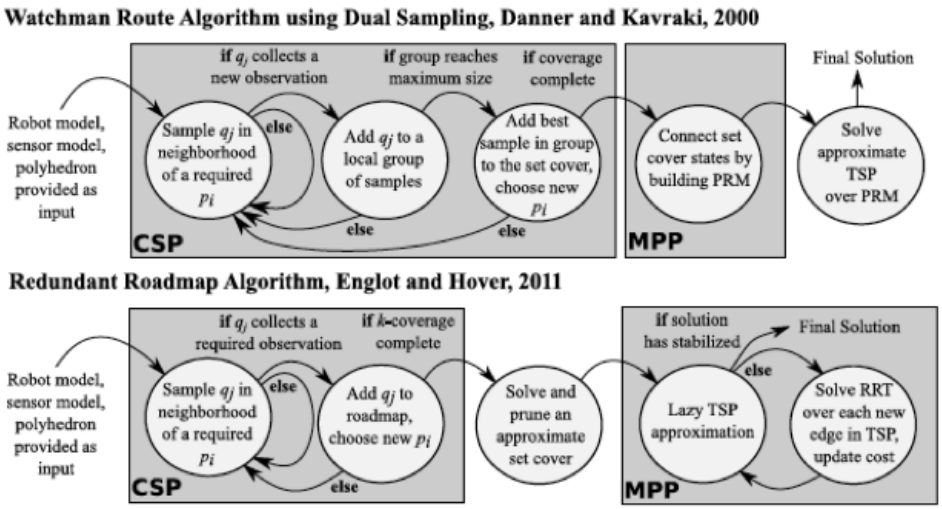


Figure 2.9: Stateflow diagram illustrating two algorithms based on CSP [16].

The watchman route algorithm using dual sampling [13] consists of a first phase in which the variation of the AGP proposed by Latombe and Gonzalez-Banos [22], [23], is used. In the second phase a weighted graph is built. This graph consists of one node for each guard, and one edge for each pair of guards with weight equal to the length of the shortest collision free path between the two nodes. To find the shortest free path between two points, Probabilistic Roadmaps (PRM) are used and an approximation of the TSP is solved over PRM.

The redundant roadmap algorithm [15] solves the variation of the AGP by randomly sampling configurations until the required structure is covered. To represent the required structure, the authors use a triangular mesh model obtained from real sonar data. The first phase lies in building a *redundancy roadmap*, that collects the robots configurations and catalogs their sensor observations, creating a discrete state space from which a inspection path will be built. In a redundancy roadmap, each point has to be covered a given number of times. So, the first phase consists in selecting a geometric primitive, that has not been covered the given number of times yet, and adding to the roadmap a random robot configuration in the neighborhood of the selected primitive.
In the second phase it is solved an approximate set cover subproblem using a

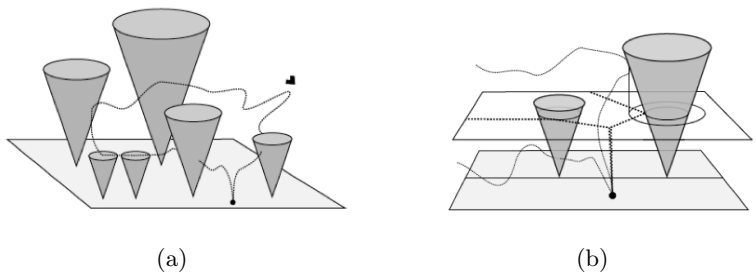
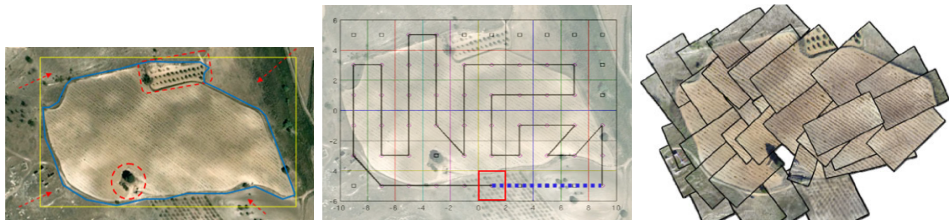


Figure 2.10: Path generated by the Cone-TSPN algorithm [33].

greedy algorithm that adds at each iteration the roadmap node with the largest set of observed primitives not yet in the cover. Then, the set covered is pruned with an iterative approach. At each iteration a configuration, that is not the unique observer of a geometric primitive, is randomly removed from the set cover until every configuration in the set is the unique observer of at least one primitive. Then, it is invoked a lazy TSP algorithm, the one proposed by Christofides [11], employing an iterate solution of the Rapidly-exploring Random Tree (RRT) over all goal-to-goal paths in the tour.

Another important study consists in the trade off between flight-time and coverage: Plonski and Isler [33] propose an offline approach based on a variation of Neighbourhoods-TSP (TSPN) called Cone-TSPN. In order to capture an image of a set of chosen points, the algorithm considers a set of inverted cones, each one with the vertex in one of the chosen points, with slope $\Pi/2 - \alpha$ and a given height $h \in H$.
The aim is to find a minimal tour that intersects all the cones. Due to the fact that this problem is NP-hard, the authors find the tour that better approximates the length of the optimal tour. Let be $\widehat{h}$ the max estimated height, defined as $\widehat{h}/2 \leq h^* \leq \widehat{h}$, where h^* is the maximum height obtained by the optimal tour.
They compare the cost of a tour on cones at the same height with the one of the tour on cones at different heights. The analysis is different for disjoint cones and for non-disjoint cones. For disjoint cones at same height a SLICE-VISIT strategy is proposed. It consists in truncating all cones higher than the $\widehat{h}$, intersecting all the cones with a plane at height h_t, where $h_t = min(H)$, and finding the TSPN tour that visits all the circular cross sections of the cones in this plane returning to the starting point at height 0. Considering different heights SLICE-VISIT cannot move higher than $min(H)$ so the authors classified the cones according to their height performing SLICE-VISIT for each class of



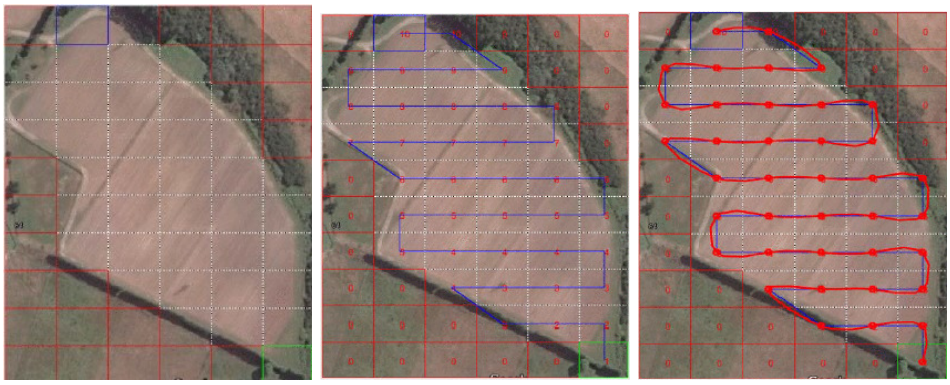
(a) (b) (c)

Figure 2.11: (a) Workplace identification [41]. (b)Coverage trajectory generated [41]. (c) Mosaic reconstruction [41].

cones, in practice they consider multiple planes. The authors obtain an approximation factors of the length of tour that is independent from the number of cones in both cases: for single height $O((\frac{max(H)}{mean(H)})^2(1+tan(\alpha)))$ and for multiple heights $O((1+log\frac{max(H)}{min(H)})(1+tan(\alpha)))$. So the tour at different heights computes tours shorter than the one at the same height. Authors extend this analyses also for non-disjoint cones. Notice that, differently from us, Plonski and Isler select points of interest a priori. Also, their problem is formulated within an Euclidean plane without obstacles.

Valente et al. in [41] apply the problem of coverage to the precision agriculture. The authors first decompose the environment, using a grid-based representation with optimal dispersion dividing the space in cubes. Then the grid is converted in a graph and the coverage path is computed selecting from the start point the nearest neighbor cell in gradient order. When more neighbors are present, a cost function is used to select the best one. Using a depth-limited search a tree of all the possible coverage paths is built in order to select the one that passes thorough all the nodes only once. The computed path is shown in Figure 2.11(b).

Nam et al. in [31] describe an offline approach for UAVs CPP in a survey mission. The method adopted by the authors uses a grid-based decomposition and a wavefront algorithm. The terrain that must be covered is divided in rectangles like in Figure 2.12(a) (start position is represented by the rectangle in blue, the green one is the goal that must be reached from the robot). It is assumed that the Field Of View of the robot can sense all the rectangle's area from the centre of it. After this a wavefront algorithm provides all the solutions from start to the goal point avoiding obstacles and the one with the best cost is selected as solution (Figure 2.12(b)). To improve the final result they implemented also a cubic spline



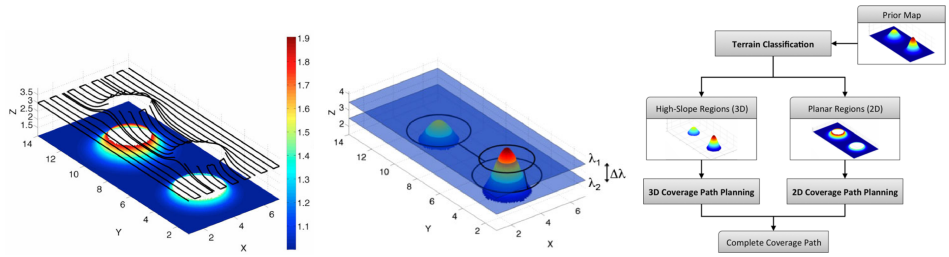
(a) (b) (c)

Figure 2.12: (a) Grid-based decomposition [31]. (b)Optimal path generated by wavefront algorithm [31]. (c) Coverage trajectory generated by cubic interpolation [31].

algorithm to smooth the path cutting away the angles as in Figure 2.12(c).

Galceran et al. in [19] propose an offline method for covering complex structures over the ocean floors using an AUV. The idea is to use a 2.5-dimensional (2.5D) bathymetric map of the floor to distinguish the planar area in the target zone and the high slopes that represent the 3D objects. Two different algorithms are then used to cover these two types of zones, producing different paths. Regarding the 2D planar surface, CPP is done by dividing the area with boustrophedon decomposition and a simple mowing-the-lawn motion producing a path similar to the one in Figure 2.13(a). For the 3D high slopes the bathymetric map is intersected with slice planes and an AUV contours the slopes maintaining a fixed offset from the target surface as we can see in Figure 2.13(b). These two distinct paths must be merged into one final path that covers the entire seabed as we can see in the tree in Figure 2.13(c). The problem to merge differents paths is not specifically addressed in the article, but is important as in our thesis we also have different paths to be merged into one.

Recently, Amigoni and Riva in [2] proposed an offline 3D CPP to cover a 2D surface using an UAV. The problem faced is to cover a field in the least possible time with a robot that starts in a fixed point and return to this. They discretized the environment in different 2D planar surface at different heights each one represented by a grid-based map. The solution adopted is a two-step approach that combines AGP to select the minimal number of points to cover the target surface and TSP that search the optimal path that passes through all the points selected



(a) (b) (c)

Figure 2.13: (a) Coverage path of the planar region [19]. (b)Coverage path planning of the high slopes [19]. (c) Diagram of the coverage path-planning algorithm for bathymetric maps. [19].

in precedence. In the next chapter this work is studied more as its the starting point for our thesis.

There are CPP works that propose also a non uniform type of coverage. These algorithms do not cover the entire target surface but they skip some area because, for some criteria (can be various and depend on the purpose of the CPP), are not important towards the goal of the CPP. Because of this the final coverage path goes only through some zones, reducing the target surface. This approach saves resources and minimizes the total cost.

Sadat et al. in [36] propose an online non uniform algorithm for CPP of a field. It can be used in a vast variety of applications such as agriculture, surveillance, search and rescue, and vegetation monitoring. The solution uses an UAV with a micro-camera whit a square FOV that has an higher resolution when its distance from the target surface is minimal. The method exploits the visit over a tree based on the Hilbert Space Filling Curve (SFC) to calculate a path minimizing the cost and exploiting the locality of the interesting regions. The tree is constructed in a way that the nodes within a depth represent a square in the map and a particular resolution (height from the target surface). Root represents the highest height while the leaves the height where the camera resolution is at its maximum. The authors use the Hilbert curve to impose an ordering on the nodes at each level of the coverage tree. The Hilbert curve is a fractal space-filling curve shown in Figure 2.14(a). Informally speaking if a robot follows a Hilbert trajectory, it is certain that it stays close to the recent places that it has visited. This means that when an interesting region is observed, one can opportunistically assume that the next node will also be interesting due to the locality preserving feature of the Hilbert curve 2.14(b). The final algorithm starts visiting the root of the tree, every

time a node is visited, decides if that is an important area and it is meaningful to go down the tree increasing the resolution. If is not an important zone, algorithm goes up in the tree.

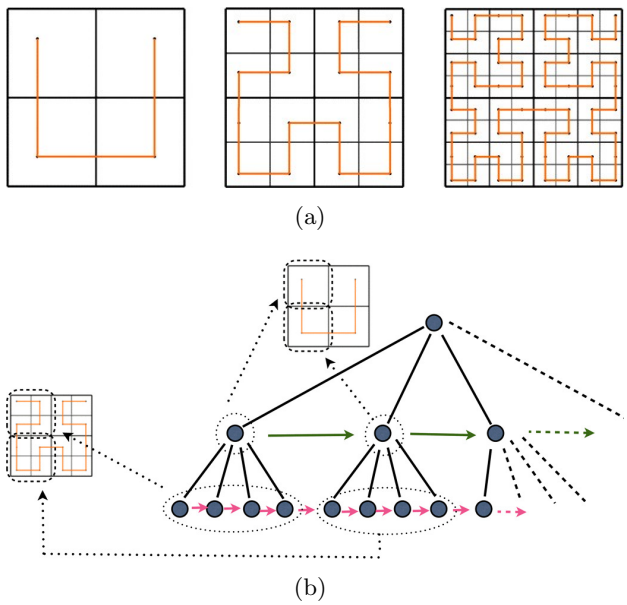


Figure 2.14: (a) Hilbert curve with different orders [36]. (b) Coverage tree with hilbert-based ordering of nodes at each depth and its relationship to grids in different resolutions [36].

Lee et al. in [29] propose an online non uniform approach to cover the seafloor using an AUV with a FOV represented by a rectangular polyhedron. The 3D environment is simplified by 2D planes at different depths. The solution relies on the concept of Artificial Island (AI). An AI is an area in a plane in which there are no obstacles and boundaries. The algorithm works in a recursive way, it consecutively scans the entire area of a plane before moving up to the next plane. Now, defining η as the sensor reliability coefficient, z_t as the measurement value, x_t as the robot position at time t and m as the environment information and h as the length of the robot sensor, if the equation $\eta \leq p(z_t \leq \frac{h}{2}|x_t, m)$ holds it means that a boundary or obstacle exists in the upper plane with respect to the current position. In the other case, when the equation does not hold it means that in the upper plane in the position where the perception is done in the lower plane, there is a safe area. Due to this, that area in the covering of the next plane can be skipped reducing the time end the cost of the final path.

Finally Table 2.2 reports all the 3D CPP methods analyzed in this section and

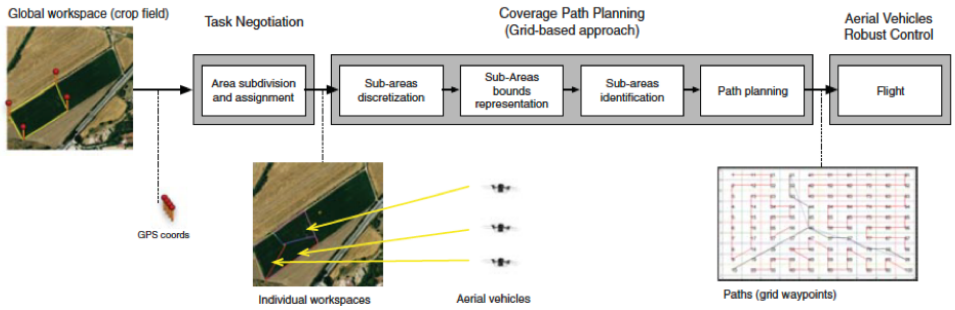


Figure 2.15: Flowchart of the system proposed by Barrientos et al. [4].

the the most important characteristics.

2.4 Multi-UAV Coverage

Regarding the use of multiple robots for CPP, the literature highlights many proposals due to the advantages of extending the CPP from single robot to multiple robots as the decrease in time, improvements in robustness, and so on. Therefore many of the works discussed above were transformed and reformulated to multi-robot coverage like Rekleits et al. [34] who extend the boustrophedon decomposition to multiple robot scenarios imposing some rules to coordinate the robots.

Maza and Ollero [30] proposed a method for multi-robot CPP at constant height. Firstly the target area is decomposed in polygonal regions using sweep line approach, then each polygon is assigned to a different UAV taking in account the capabilities of each UAV, like flight endurance and range. Once each UAV has an area assigned, this is covered by a back and forth motion along rows perpendicular to the sweep direction of the polygon so as to ensure the minimum number of turns in an area.

A multi-UAV approach at constant height is proposed by Barrientos et al. [4]. The aim of their work is to propose an efficient algorithm for precision agriculture using a multi-UAV system. Initially the target space is divided in different areas solving a task subdivision and allocation problem with two restrictions: each robot knows its own characteristics and status but does not know anything about the other robots. They solve this problem as a negotiation process in which each robot tries to obtain as much area as possible, rather than assign a region to a robot using geometric considerations. Then the path to cover each waypoint

Reference	Algorithm	Offline Online	Trajectory	Uniform Non-uniform
[26]	Multiple 2D planes at different heights Semi-approximate cellular decomposition	Online	Fixed (zigzag along parallel lines)	Uniform
[9]	Geometric semplification of target complex structures	Offline	Fixed (emisphere)	Non-uniform
[6]	AGP+TSP	Offline	Computed	Uniform
[13]	Watchman route algorithm with dual sampling (variation of AGP based on the idea of CSP + weighted graph)	Offline	Computed	Uniform
[15]	Redundant roadmap algortihm (variation of AGP based on the idea of CSP + set cover sub-problem + lazy-TSP)	Offline	Computed	Uniform
[33]	Cone-TSPN at different heights (variation of Neighbourhoods-TSP)	Offline	Computed	Uniform
[41]	Grid-based decomposition in cubes Graph tree visit	Offline	Computed	Uniform
[31]	Grid-based decomposition Wavefron algorithm (rectangles) Cubic spline to smooth the path	Offline	Computed	Uniform
[19]	2.5D bathymetric map Merge of two different algorithms (boustrophedon decomposition for planar area, 3D countour for slopes)	Offline	Fixed (mowing-the-lawn, circles for slopes)	Uniform
[2]	2D planar surfaces at different heights AGP+TSP	Offline	Computed	Uniform
[36]	Grid based Visit of a tree exploiting Hilbert Space Filling Curve	Online	Computed	Non-uniform
[29]	2D planar surfaces at different heights Covering consecutively planes exploiting AIs	Online	Depends on coverage algorithm used	Non-uniform

Table 2.2: Summary of 3D works presented.

(each point of interest) has to be planned for each robot and it can be solved as a simple CPP: they have developed an extension of the wavefront planner previously explained. A scheme of this process is shown in Figure 2.15.

The two previous papers consider only one height of flight for UAVs, while Basilico and Carpin [5] propose a multi-UAV CPP on two levels for surveillance. They consider heterogeneous UAVs and divide them in *sentinels* and *searchers*, where sentinels are positioned at higher heights and are tasked to control large areas and detect some type of events called attacks. Searchers depend on sentinels, since they fly at lower heights and detect with less errors an attack when notified by the sentinels. The authors model the environment as a grid of equally sized squared cells and the presence of an attack in a cell c as a loss value $l(c)$. They define a binary function $a(c,t)$ that indicates if a cell c is attacked at time t. Their goal is to minimize the overall loss computed as $\sum_{c \in G} l(c) \int_0^T a(c,t)dt$, where G is the search domain, and T is a finite time horizon. Their aim is to cover all the area at risk, so if a zone is safe, no UAV will cover it.

Finally Table 2.3 reports all the multi-robot CPP methods analyzed in this section.

Reference	Algorithm	Online Offline	Trajectory
[34]	Extension of 2D boustrophedon decomposition to multiple robots	Offline	Fixed
[30]	Polygonal regions decomposition (each polygon assigned to a root)	Offline	Fixed (back and forth)
[4]	Division into sub regions (negotiation between robots to obtain the most wide area)	Offline	Depends on the CPP used
[5]	2 levels division UAVs on the upper level cover large areas UAVs on the lower level cover little areas	Offline	Computed

Table 2.3: Summary of multi-robot works presented.

Chapter 3

Problem Setting

In this chapter we formalize the problem of our studies. We will describe the work done by Amigoni and Riva in [2] and by Greta Ghiotti in [21] since it is the starting point for our considerations. Then, we introduce our idea to improve the results obtained in these two works.

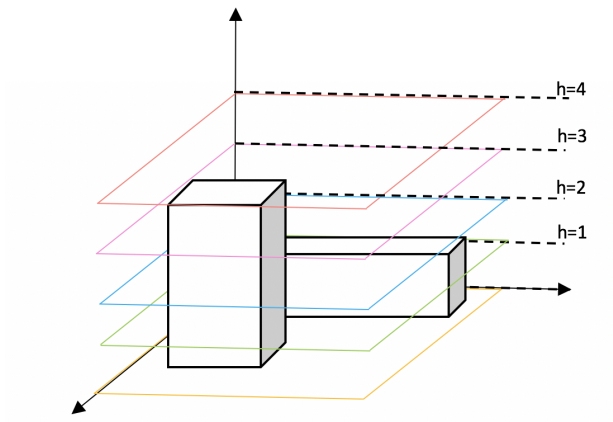
3.1 Problem Statement

In [2] the authors presented the coverage path planning of a robot that moves in a 3D space and that must cover a 2D target surface (a field for example).
They considered a single robot that can move in a three-dimensional environment as an Unmanned Aerial Vehicle (UAV) starting from a fixed cell (called depot) and returning to the same depot. The robot has to compute a tour such that the ground of the environment is completely covered by the robot's sensor at the minimum cost (either time or distance travelled).

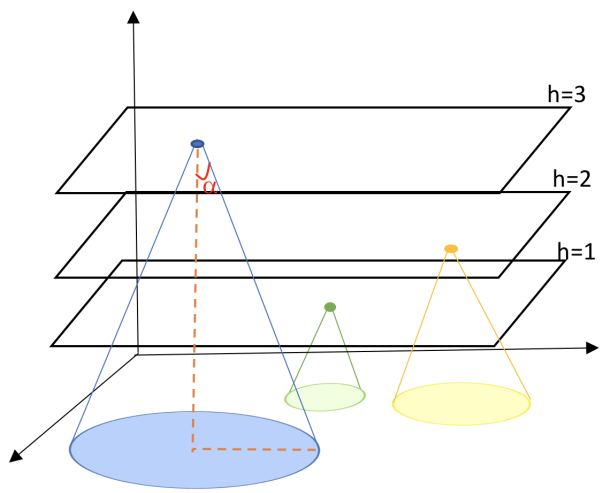
The environment is discretized by 2D planes at different heights and each plane is modelled as a grid of identical cells with a square shape. We can see this model in the Figure 3.1(a).

Each cell is defined by a set of coordinates (x, y, z) where x,y denote position in the plane, while z denotes the height in space. Each cell can be *free* if the UAV can move in it or *occupied* if there is an obstacle that prevents movement. The depot is always on the ground and is defined by $(x, y, 0)$.

The UAV has a sensor with a conic Field Of View (FOV) that projects to the ground a circle with a radius equal to $r = h \cdot \tan(\alpha)$, where h is the height of cone and α is half of the apex angle (Figure 3.1(b)). It is assumed that each perception $P(f)$ (where f is a *free* cell) is constant and is equal for any $f \in F$ for every height.



(a)



(b)

Figure 3.1: (a) Example of 3D decomposition in different planes [2]. (b) Example of the FOV varying with height [2].

The goal of the CPP presented was to find a path that covers all the ground cells that is,

$$\bigcup_{1<i<k} C(f_i) = F(0).$$

where $C(f_i)$ represent all the area covered at the ground from the cell f_i, $F(0)$ is the set of *free* cells at the ground and k is the length of a generic path. The cost

of a solution is a sum of all the distances to travel from point to point and the perceptions cost which is a scalar,

$$c(T) = \sum_{i=2}^{k} c(f_{i-1}, f_i) + \sum_{i=2}^{k-1} P(f_i). \tag{3.1}$$

They classified the environments in two possible types:

1. type 1: an open field (like an agricultural one), where the coverage function is monotone with respect to the height. In this type of environment the UAV will cover more area when going up. This is true when it holds the following equation,

   $$C(f) \subseteq C(f'), \qquad 0 \leq h < h' \leq H_{max}.$$

   where $C(f)$ and $C(f')$ are respectively the coverage function at $f = (x, y, h)$ and $f' = (x, y, h')$. An example of this environment can be seen in Figure 3.2(a).

2. type 2: a field where there are open building (like caves). In this case the area that can be covered at a given height is a subset of the area covered at any lower height,

   $$\bigcup_{f \in F(h')} C(f) \subseteq \bigcup_{f \in F(h)} C(f), \qquad 0 \leq h < h' \leq H_{max}.$$

   Figure 3.2(b) shows an example of this environment.

In environments of type 1 they demonstrated that the space of the solutions can be reduced to only tours over a single fixed height without worsening too much the optimal solution. Results in this case showed that the optimal tours in terms of path length are at the highest height with a little FOV apex angle because this will reduce the number of covering points. When the FOV apex angle gets bigger the best paths are in the middle heights because when the UAV has a very large FOV there is no advantage in going up due to the fact that after a certain height there is not a reduction in the number of covering points. Going at higher levels in these cases is only counterproductive because of the travel cost of going up.

In type 2 environments was invented a coverage tour over multiple heights. It is assumed that only the cells at height K can cover all the target cells at h_0 because the openings in obstacles and buildings can occur only at height K (assuming that the openings are only at $h_{min} \neq h_0$). The coverage algorithm calculates paths over the combinations of height K (fixed) and a level $h \in (H \setminus h_0)$

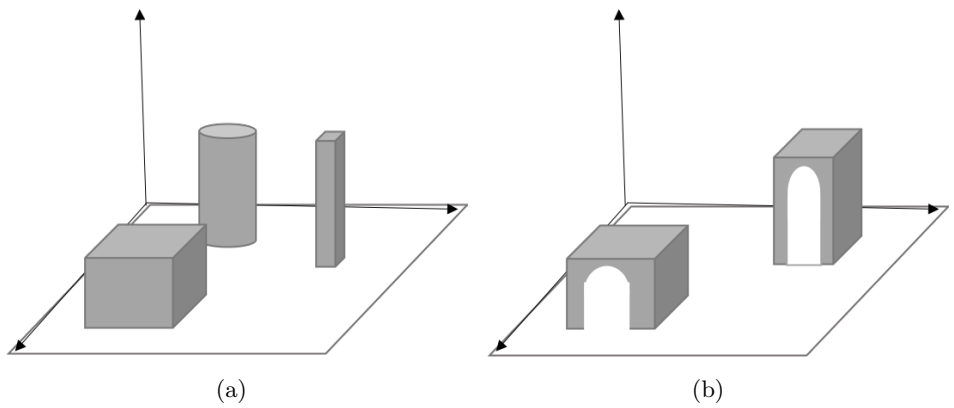


Figure 3.2: Examples of (a) Type 1 (b) Type 2

where H is the set of all the heights. Firstly an *ArtGallery* procedure finds the covering points at height h, then *ArtGallery* is called again at height K to cover the cells at the ground that can not be covered from height h. From height K is assumed that all the cells at the ground can be covered. Lastly, a TSP procedure is called to find a path between the covering points that minimizes the distance cost. In this case results showed that a path over a single height is never preferred over a path over 2-height because like in case 1 going up can reduce the number of the covering points, reducing the total cost too.

Regarding computational cost the algorithms with the most significant impact are the AGP and TSP. Results showed that AGP time depends on the extension of the area to cover (larger is the area bigger is the time to compute AGP) while TSP depends on the number of covering points ($TSP_t = O(n)$).

3.2 Problem Analysis

The problem that we face in this study is to obtain a better computational cost of the coverage algorithm presented in the previous section.

Our approach is a divide et impera process. In computer science a divide et impera approach consists in simplyfing the problem in two or more sub-problems that are more simple to resolve. After that, the solutions of the sub-problems must be combined together to produce the final result.
In our case, we divide the entire map into different 3D-zones producing different paths that must be merged into one unique path to obtain the coverage path of

the entire environment.

A 3D-zone is a 3D portion of the entire map such that the set of cells contained in the 3D-zone at the ground is equal to the set of cells contained in the 3D-zone at the other heights, in other terms the area of the zone in all the heights must be equal to the area of the zone at the ground. Also, we assume that each sub-zone does not intersect with a different one and that the union of all the sub-zones is equal to the entire map,

$$x_i \cap x_j = 0, \qquad x_i \cup x_j = X \qquad \forall j \neq i$$

where X is the entire 3D map and x_i and $i = 1, ..., n$ represents the 3D-zones in which is divided X.

The division of the map is done such that a 3D-zone have the characteristics of only one environment (type 1 or type 2) and is not a mixed type environment. By doing this, we can cover the ground with SingleHeight algorithm if the portion of the map is type 1, or TwoHeight algorithm if the portion of the map has the characteristics of type 2 maps. In Figure 3.3 we can see a simple environment division into two distinct 3D-zones of different types. In the end we will have a path (starting from a common depot and returning to it) for every 3D-zone in which the entire map is divided.

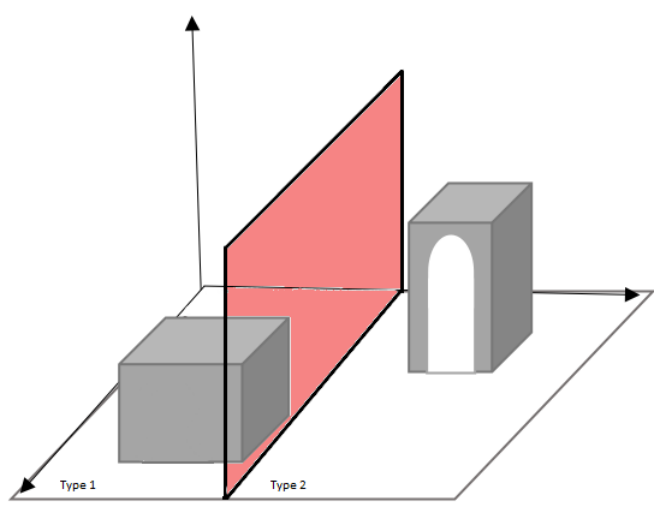


Figure 3.3: Division of the entire map in two 3D-zones. On the left a type 1 environment, on the right a type 2 one.

All these paths must be merged into one unique path minimizing its distance

cost. The resulting path will have the same depot of the paths merged and a number of covering points equal to the sum of covering points of the two paths excluding the depot for one of them. A similar process is done in [19] and is described in the previous chapter.

Using this divide et impera approach, we have a reduction in terms of computational time because AGP has to work on a little portion of the target area, and the number of covering points obtained with AGP is limited, especially with larger FOVs, helping TSP to obtain a path in less time with respect to obtaining a path over the entire environment.

To this possible reduction of computational time corresponds a possible worsening in the distance cost because the TSPs over the zones are optimal only over the area covered. In fact merging paths that have TSP local optimality over a 3D-zone does not produce an optimal path over the entire area.

Chapter 4

Algorithms

In this chapter we describe all the algorithms used in our 2-step approach. Firstly we present the methods that are used to calculate the covering paths in the 2 different types of environments introduced in Chapter 3. Then we describe two different merging algorithms.

4.1 First step process methods

In this section we analyse the methods used to find the covering points through AGP and the ones used to produce a path with TSP.

4.1.1 Art Gallery Problem

The AGP is the very first step of our entire process. AGP represent a visibility problem born from a real-world problem of guarding an art gallery with the minimum number of guards. From this, AGP was studied a lot during the years and is defined as a minimum number of guards in a environment, which together can view the whole environment. This problem is NP-hard also in simple environments.

We solve the AGP as a set cover problem, so our aim is to find the smallest possible number of cells, called S, that together can cover all the target cells on the ground $F(0)$ (in our case F_0 represent the ground of the zone on which the AGP is called). The set cover problem is again a NP-hard problem, but in [38] the authors demonstrated that the number of covering points found by general greedy approaches is, at most, $OPT \cdot log|F(0)|$, where OPT is the number of covering points of the optimal solution of the AGP.
We propose a greedy algorithm, choosing at each iteration the guards with the highest number of cells covered among the remaining uncovered target cells at the

ground (h_0). When more than one guards cover the same number of cells, one is selected randomly. After a covering point is selected, it is removed from the set of possible covering points and the covered cells are removed from the target cells. Therefore, the next covering point is selected from those that cover the remaining uncovered cells. The algorithm finishes when all the target cells are covered. AGP is described in Algorithm 1.

```
Function ArtGallery(h, F(0)) is
    while isempty(F(0)) do
        s = arg max(CoverageArea(h, F(0)))
        S.Insert(s)
        F(0) = F(0) \ C(s)
    end
    return S
end
Function CoverageArea(h, F(0)) is
    return the number of cells at the ground of the 3D-zone covered by
      each cell at height h in the environment
end
```

Algorithm 1: Art Gallery function

In the AGP algorithm, the *CoverageArea* function will return the candidate covering cells for our paths. In our case, the candidate covering cells are selected among the cells within the borders of the 3D-zone on which the AGP is called. To obtain the set of these cells we manually set the boundaries of the 3D-zone that we are considering and a simple algorithm selects all the cells in these borders. In our solution, we must produce a set of covering points through AGP for every 3D-zone in which is divided the entire environment.

4.1.2 A* and Theta* algorithms

We use Theta* search algorithm to compute the distances between covering points before applying the TSP and after to obtain the complete path between the covering points. Since Theta* is based on A* algorithm we quickly introduce this before speaking of Theta*.

A* algorithm

The A* algorithm [25] is an extension of Dijkstra algorithm. A* uses a heuristic function $h(s)$ to choose a neighbour vertex without exploring all the neighbour vertices. This heuristic function is an estimation of the cost from the current vertex s to the goal. A* maintains two values for every vertex: the g-value $g(s)$

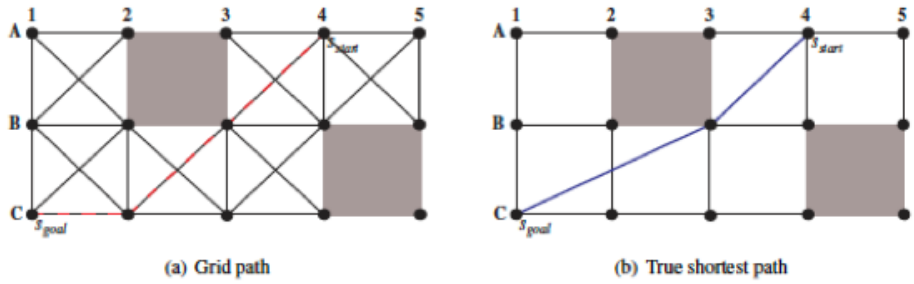


Figure 4.1: A grid path versus true shortest path [12].*

that is the length of the shortest path from the start to the current vertex, and the parent value $parent(s)$. A* uses two sets: *open* and *closed* to evaluate the vertices.

- The *open* set is a priority queue (based on the g-value $g(s)$ of every vertex plus the heuristic function $h(s)$, $g(s) = g(s) + h(s)$), that contains the discovered vertices that are not evaluated yet.
- The *closed* list is the set of vertices already evaluated.

A* uses the $UpdateVertex(s, s')$ function to update the g and *parent* values of an unexpanded visible neighbor s' of s using the procedure $ComputeCost(s, s')$. $ComputeCost$ updates the g and *parent* values only if the sum between the actual distance from the start to the current vertex $g(s)$ and the cost of the path in straight line between s and s', $c(s, s')$, is lower than the actual distance between s' and start, $g(s')$. When the goal is reached, the algorithm ends. The procedure is reported in Algorithm **??**.

If $h(s)$ is admissible (it never overestimates the cost of reaching the goal) and consistent (its estimate is always less than or equal to the estimated distance from any neighboring vertex to the goal, plus the cost of reaching that neighbor.), A* guarantees to find the shortest path on a graph restricted to the grid shape, but this shortest path does not correspond to the shortest path in the continuous environment as shown in Figure 4.1. For this reason Theta* algorithm was developed firstly in [12] and used today in many covering problems like ours.

Theta* algorithm

Theta* was firstly introduced by Daniel et al. in [12]. The main difference between

Theta* and A* is in the *ComputeCost* function where two possible paths can be found: Path 1 as before is the path in straight line, instead, Path 2 considers also the path from the start vertex to $parent(s)$ and from $parent(s)$ to s', if s' has line-of-sight to $parent(s)$. This allows the parent of a vertex to be any vertex. Again the g and *parent* value of s' are updated only if the sum of the $g(parent(s))$ and $c(parent(s), s')$ is lower than the shortest path from the start vertex to s' found so far. The new pseudocode of *ComputeCost* function is described in Algorithm 2.

```
Function ComputeCost(s, s')is
    if LineOfSight(parent(s),s') then
        // Path 2
        if g(parent(s)) + c(parent(s), s') < g(s') then
            parent(s')=parent(s)
            g(s')= g(parent(s)) + c(parent(s),s')
        end
    else
        // Path 1
        if g(s) + c(s, s') < g(s') then
            parent(s') = s
            g(s') = g(s) + c(s, s')
        end
    end
end
```

Algorithm 2: Pseudocode for Theta* algorithm [12].

Distances with Theta*

In our problem Theta* is used not only to calculate the final path between all the covering points but also to calculate the matrix distances that is used by TSP to find the optimal order of visit. To do this, a variation of Theta* is used. The goal is removed and the algorithm is iterated for each coverage point returned by AGP. Substituting at each iteration the starting vertex with the current coverage point we calculate the g-value that corresponds to the distance between the pair of points considered. When all the points are visited the distance matrix is generated and passed to the TSP.

4.1.3 Travelling Salesman Problem

Once the covering points are selected, it is necessary to find an order to visit them. The TSP returns the shortest tour that visits each covering point and go back to the starting one, given in input the set of covering points and distances between them. Like the AGP it is a NP-hard problem, and its complexity increases with the number of covering points.

It was formulated as an integer program for the first time by Dantzig, Fulkerson, and Johnson [14]. In our coverage problem we follow their formulation.
Once the AGP has returned the selected points S, we want to minimize the travelling distance between them.
Before applying TSP, we transform the 3D environment in a weighted graph G, where the vertex of G are the cells of the environment and the edges have a weight equal to the distance d_{ij} between each pair of vertex i and j of G. d_{ij} is calculated using Theta* (explained in Section 4.1.2).
Let V be the set of vertices of the graph G. Then the total travelled distance is the sum of the distances of the vertices in the tour:

$$travelledDistance = \sum_{(i,j)\in V} d_{i,j} x_{i,j}.$$

Where $x_{i,j}$ is a binary variable, that is 1 if $cell(i,j) \in tour$, 0 otherwise.
The tour should pass only once through each vertex:

$$\sum_{j\in V} x_{i,j} = 2, \forall i \in V.$$

Therefore for every vertex i exactly two of the associated $x_{i,j}$ variables should be equal to 1 so that the condition above is true, meaning that a vertex x_i can appear only one time in a path (one, when from a generic cell we go to the cell x_i and two, when from x_i we go to another cell). At this moment we can produce solutions that are not connected tour, finding subtours. So we add a constraint to eliminate subtours, requiring that for each nonempty subset Sub of the set of covering points V, the number of edges between the vertices of Sub must be at most $|Sub| - 1$, so:

$$\sum_{i,j\in V, i\neq j} x_{i,j} \leq |Sub| - 1, \forall Sub \subset V, Sub \neq 0.$$

So finally the integer program becomes:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in E} d_{i,j} x_{i,j} \\
\text{subject to} \quad & \sum_{j\in V} x_{i,j} = 2 && \forall i \in V \\
& \sum_{i,j\in Sub, i\neq j} x_{i,j} \leq |Sub| - 1, \quad \forall Sub \subset V, Sub \neq 0 \\
& x_{i,j} \in \{0,1\}
\end{aligned}
$$

This is only one of the possible methods that can be used to produce a solution to a TSP problem, in literature there lot more of this for example Neighbourhoods-TSP (TSPN) described in Chapter 2.

4.2 The Coverage Algorithms

In this section we describe how the two-phase method previously proposed is applied to the environments of Case 1 and Case 2 presented in Chapter 3.
Type 1 comprises all the environments with a monotone behavior, as the case of covering a field. Basically we cannot have an obstacle at height h if this obstacle is not present at height $h' < h$.
Instead, Type 2 includes the environments in which the area that can be covered at a given height is a subset of the area covered at any lower heights.
We decided to formulate two different algorithms for the two cases:

- SingleHeight algorithm for Type 1,
- MultiHeights algorithm for Type 2.

4.2.1 SingleHeight Algorithm

SingleHeight Algorithm is used to cover environments of Type 1 described in Chapter 3. As we have seen, to cover this type of area it is sufficient to generate a path over each 2D plane independently with AGP and TSP and then select the path with the minor distance cost. This is possible because we are sure that if we have not an obstacle at height h we are sure that an obstacle is not present at height $h' < h$ (with $(x, y) = (x', y')$)
As shown in Algorithm 3, the SingleHeight algorithm considers each of the 2D grids at the different heights $h \in (H \setminus h_0)$. Firstly, the $ArtGallery$ procedure is applied to every grid to find the covering points. Then $Theta^*$ is called to calculate the distances between each pair of covering points. Eventually, the TSP algorithm is applied $|H| - 1$ times every time using the covering points of the 2D

grid considered. This means that we obtain $|H|-1$ different tours, each one with a different cost, for every height $h \in (H \setminus h_0)$. The SingleHeight algorithm selects the tour with the minimum cost.

```
Function SingleHeight(F(0),H)is
    for h ∈ (H \ h_0) do
        tour_h = 2phases(h,F(0))
        if First Iteration then
            tour=tour_h
        else
            if c(tour_h) ≤ c(tour) then
                tour= tour_h
            end
        end
    end
    return tour
end
Function 2phases(h, F(0))is
    S = ArtGallery(h,F(0))
    D = Theta*(S)
    tour = TSP(S,D)
    return tour
end
```

Algorithm 3: Pseudocode for SingleHeight algorithm [2].

4.2.2 TwoHeights Algorithm

In Case 2 the situation is different, and we must consider all the possible tours at multiple heights. We assume that given a set of discrete heights H, for only one height $K \in (H \setminus h_0) : \bigcup_{f \in F(K)} C(f) = F(0)$ that means that only the cells at height K can cover the target cell at the ground. K is the minimum height excluding the ground.
Hence, all the possible combinations of heights $h \in (H \setminus h_0)$ with a fixed element that is height K must be considered to find the minimum covering tour.
Initially, the TwoHeights algorithm computes the 3D graph G. Then each pair of heights K, h with $h \neq K$ is considered one at a time: firstly, the covering points are found with the *ArtGallery* procedure at h. After, the algorithm computes the uncovered cells at the ground and the *ArtGallery* at height K is called in order to find the guards of the remaining uncovered cells at the ground. Later, the distances between each pair of covering points are calculated using Theta*. At the end, the TSP is applied to the covering points and their distances. When

the costs of all the tours are derived, the best combination of two heights is chosen and returned.

```
Function TwoHeights(F(0),H)is
    for h ∈ (H \ h_0) do
        S = ArtGallery(h,F(0))
        // the already covered cells are removed
        U = F(0) \ C(S)
        S = S ∪ ArtGallery(K,U)
        D = Theta*(S)
        tour_h = TSP(S,D)
        if First Iteration then
            tour=tour_h
        else
            if c(tour_h) ≤ c(tour) then
                tour= tour_h
            end
        end
    end
    return tour
end
```

Algorithm 4: Pseudocode for TwoHeights algorithm [2].

4.3 Second step process - Merge algorithms

In this section we describe two different algorithms to merge different paths. Both the algorithms are used to merge the paths greedly over all the heights. This means that having two different zones, the $path_1$ at height $h \in H \setminus h_0$ is merged to the $paths_2$ of all the heights $H \setminus h_0$.
Our algorithms do the merge operation between two paths but can be extended to merge more paths by simply merging the paths 2-by-2.

4.3.1 First Merge Algorithm

This is a plain algorithm and is the most simple and is faster than the other one in terms of computational times.
The main idea is to go select a path as first and a path as second. The sequence of the final path will be all the points of the first path until the last one (excludind the depot at the end) and then all the points of the second path (excluding the depot at the beginning). With this method two edges are eliminated and one new

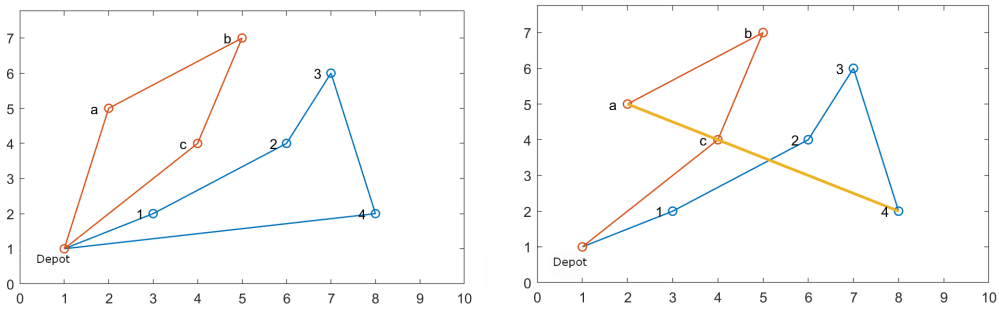
link is added. To decide which of the two paths goes first a control is done (see Algorithm 5).
The order of the covering points is maintained through the merge because the new path obtained goes fully through one path before to go to the other one.
An example of two paths merged can be seen in Figure 4.2. In this case the path in blue is selected as first path (it maintain its first edge and loses the last one), while the red path is the second (loses first edge and maintain the last one). Edge in yellow is the new edge that connects the last point of the path 1 to the first point of path 2.

```
Function FirstMergeAlgorithm()is
    for h ∈ H \ h_0 do
        path_1 = path_1 at height h
        for h' ∈ H \ h'_0 do
            path_2 = path_2 at height h'
            cost, path_{h,h'} = MergePath(path_1,path_2)
        end
    end
    // select the path with minor cost and return it
end
Function MergePath1(path_1, path_2)is
    eliminate = lastEdge_1 and firstEdge_2
    if lastEdge_1 + firstEdge_2 < firstEdge_1 + lastEdge_2 then
        // swap the two paths (path_1 becomes path_2 and
           viceversa)
        // eliminate = firstEdge_1 and lastEdge_2
    end
    lastNode_1 = path_1[end] \ depot
    firstNode_2 = path_2[1] \ depot
    newEdge = Theta*(lastNode_1,firstNode_2)
    newCost = costPath_1 + costPath_2 − eliminate + newEdge
end
```

Algorithm 5: Pseudocode for the first Merge algorithm.

4.3.2 Second Merge Algorithm

This algorithm is more complex than the previous one. The idea is to choose a path as *external* and the other as *internal*. The *external* maintains the depot with its first and last edges. The *internal* becomes a closed polygon, loses its depot and the edges connected to them (first and last edges of the initial path) gains an edge from the last point to the first one.
An example of this merging technique can be seen in Figure 4.3. Blue path is the *external* one, while red with the yellow closing edge is *internal*. In purple the new



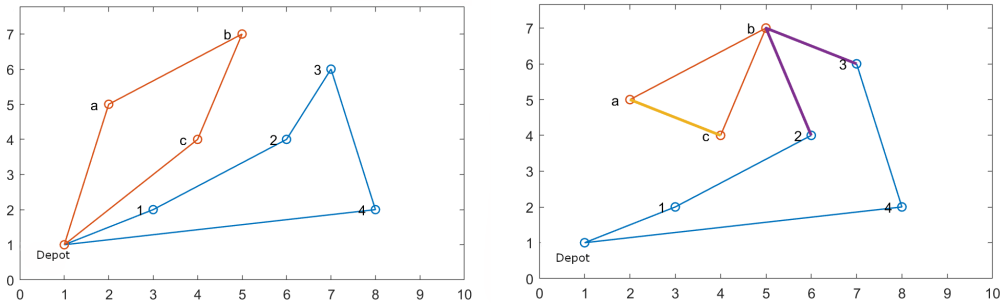
(a) (b)

Figure 4.2: Examples of (a) Two different paths (b) Paths merged using the first algorithm

edges using the strategy 1.
The algorithm iterates over all the possible pair of points of path 1 and path 2 (such as p_1 and p_2 where the former is a point in path 1 and the latter a point in path 2) to retrieve the path with the least cost possible.
When, after going through the *internal* path, we must return to the *external* one two strategies are considered and the one with the lesser cost is choosed as we can see in Algorithm 6.
Generally to search for the optimal solution in terms of cost distance the algorithm runs two times changing *internal* path with the *external* one and then chooses the result with less cost. For a particular type of map division this in not mandatory (as we will see in Chapter 5) because the optimal results will always be with one path as *internal* and the other as *external*.

```
Function SecondMergeAlgorithm()is
    for h ∈ H \ h_0 do
        path_1 = path_1 at height h
        for h' ∈ H \ h'_0 do
            path_2 = path_2 at height h'
            cost, path_{h,h'} = MergePath(path_1,path_2)
        end
    end
    // select the path with minor cost and return it
end
Function MergePath2(path_1, path_2)is
    external = path_1
    internal = path_2 + // modifications to transform the path in
        a closed polygon
    for c_i ∈ external \ depot do
        for c'_j ∈ internal do
            newEdge1 = Theta*(c_i, c'_j)
            if Theta*(c'_j, c_{i+1}) ≤ Theta*(c'_{j-1}, c_{i+1}) then
                // strategy1
                newEdge2 = Theta*(c'_j, c_{i+1})
            else
                // strategy2
                newEdge2 = Theta*(c'_{j-1}, c_{i+1})
            end
            newCost = // calculates the new cost eliminating
                edges non used and adding new edges
        end
    end
end
```

Algorithm 6: Pseudocode for the second Merge algorithm.



(a) (b)

Figure 4.3: Examples of (a) Two different paths (b) Paths merged using the second algorithm with strategy 1

Chapter 5

Experiments

In this chapter we are going to focus on the experimental results of the algorithms explained in Chapter 4. In the first part of the chapter, we are going to concentrate on the tools used and on the images used to simulate environments on which we used our CPP technique. Then, in the second part we will report some results and we will compare the different covering tours, their costs, and computational times of the tours obtained with the merge algorithm with respect to the tours over the entire environment.

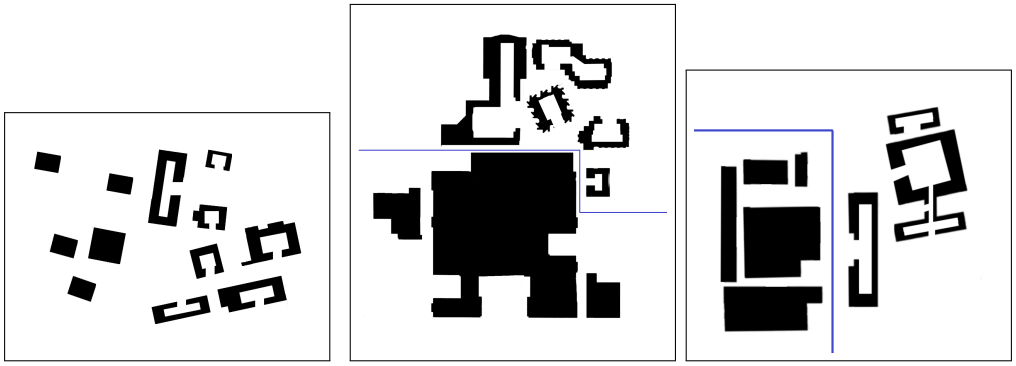
5.1 Tools

We implement our algorithms in MATLAB. We run MATLAB on a machine with a 2.4 GHz Intel Core i5 CPU and a RAM of 8GB .
As said in the previous chapters we have to solve TSP, for this reason we use an optimization tool. We use a MILP solver: GUROBI [24], a free optimization tool that works with MATLAB.
We implement all the algorithms on the basis of the pseudocode reported in Chapter 4.

We suppose to work with two different types of UAVs:

- one equipped with 1.7 mm lens, that has approximate 170 degree FOV, so, since we consider half apex angle, we will refer to it as $\alpha = 85°$;
- the other equipped with 2.8 mm lens, that has approximate 120 degree FOV, so, since we consider half apex angle, we will refer to it as $\alpha = 60°$.

Both the UAVs have a fixed camera. So, we execute every algorithm two times, each time considering a different FOV.



(a) (b) (c)

Figure 5.1: The 3 target regions at the ground used during the experiments.

We feed our algorithms with occupancy grid maps derived from the images representing the 2D grids in which the 3D environments are decomposed. To reduce the computational time we consider occupancy grid maps with maximum size of 100×100 (cells), which is the size of the corresponding matrix in MATLAB. All the occupancy grid maps with size larger than this threshold are scaled by a factor 0.1 using subsampling. For instance, if we consider an occupancy grid map of size 600×700, it is scaled by a factor 0.1. This means that each submatrix of size 10 of the occupancy grid map is replaced by one, if the number of ones in the submatrix is bigger than the number of zeros, by zero otherwise.
Occupancy grid maps with size bigger than 1000×1000 are not considered in our experiment. These limitations only have the aim of reducing the computational time, although our algorithms can work with images of every size.

We build our 3D environments, concatenating along the third dimension the 2D grids at different heights.
About the environment we only have one constraint valid for every algorithm: any cell in our 2D grids can be reached from any other cell. Practically, we cannot have white spaces closed by a black line without entrance (for example: closed yards).

5.2 Environments

We choose 3 different environments, which target area (h_0) is shown in Figure 5.1. During this chapter we will refer to the environment in Figure 5.1(a) as *Environment A*, to the one in Figure 5.1(b) as *Environment B*, and to the one in Figure 5.1(c) as *Environment C*. All the environments are taken from real maps,

Environment A is from the map of McKenna mout site [37], Environment B is from the map of the Tech Institute of Northwestern University [32], and Environment C from the map of Bovisa Campus of Politecnico di Milano [7], but they have been modified in order to create distinct zones of different types as explained in Chapter 3.
Table 5.1 reports the sizes of the 2D grids in every environment.

Environment	Matrix size 2D grid
Environment A	60×70
Environment B	80×70
Environment C	63×71

Table 5.1: Matrix size of each 2D occupancy grid map.

We discretize our 3D environments at six heights $H = \{h_0, h_1, h_2, h_3, h_4, h_5\}$. Each height corresponds to a horizontal cross section taken every two meters, so $H = \{0, 2, 4, 6, 8, 10\}$ m. In our environments the grid map at h_1 is equal to the grid map at h_0, that represents the ground. For each environment we consider 6 images that are the result of the intersection between a plane at height $h_i \in H$ and the 3D environment (Figure 5.2). These images correspond to the occupancy grid maps at a given h of the considered 3D environment, a black cell corresponds to an obstacle, while a white one corresponds to a free area in which the UAV can move.
As said in previous chapters we have two possible scenarios: Case 1 and Case 2.

For our experiments we manually divide the three environments in distinct areas that are of Type 1 or Type 2. Every map id divided in two zones, one of type 1 and one of type 2. We can see the division used in Figure 5.3 that shows the only the ground plane (h_0) because as we said in Chapter 3 the division is equal for all the planes. In Appendix A we can see how the maps evolve vertically and how the obstacles change at different heights.

5.3 Results

In this section we will illustrate the results of the two merge algorithms compared also with the results obtained without map division. The results studied are in term of distance cost and computational time.

The term *configuration* is used to indicate the environment and FOV. The environment can be Environment A, Environment B, and Environment C. The FOV can be $\alpha = 60°$ or $\alpha = 85°$. For instance, a configuration is Environment A with $\alpha = 60°$.

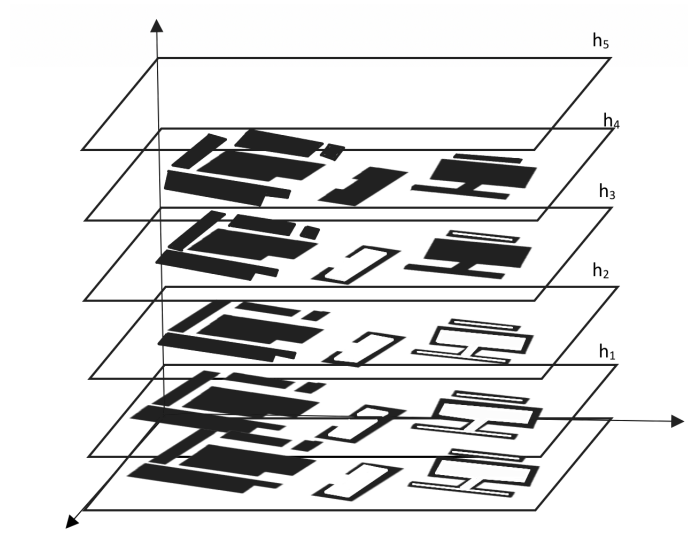


Figure 5.2: Decomposition of a 3D environment in six grids of Environment C in Case 1.

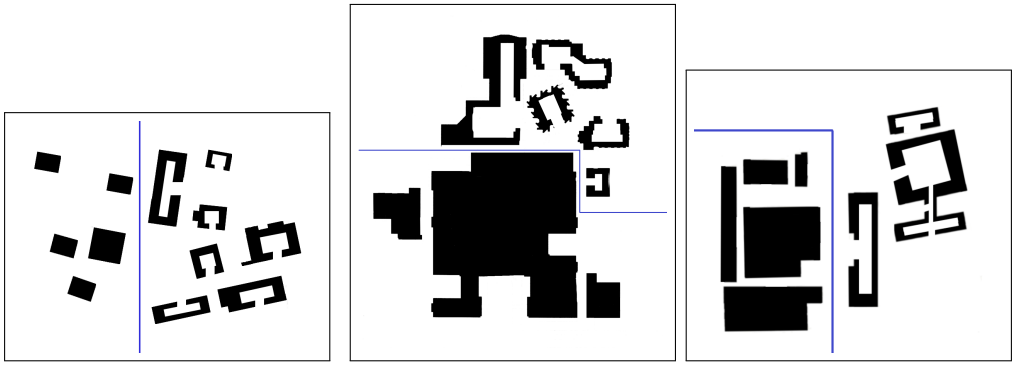
The images of the paths merged are arranged in ascending order according to height.
They are coloured using a grey-scale:

- lighter shades of gray for higher obstacles,
- darker shades of grey for lower obstacles,
- with white for the free area in every level,
- black for the obstacle present at the height of the tour.

Furthermore, we use different colours and notation to represent the total tour. More precisely:

- a full circle represents the beginning of a continuous path at a given height (with continuous path we refer to a path in the same plane, without changing height),
- a diamond indicates the end of a continuous path at a given height (represents a change in the height),
- an empty circle represents a covering point and a number indicates the order of exploration of the tour.

In all the environments the depot is situated in point $d = (1, 1, 0)$, so d is the point (1,1) on the ground (h_0).



(a) (b) (c)

Figure 5.3: Target ground division of the Environment A,B,C.

5.3.1 Environment A

Environment A is divided in two vertical sections by a line in the middle such that area on the left is of type 1 and covered by SingleHeight algorithm and the area on the right is covered by a TwoHeight algorithm because is of type 2 (see Figure 5.3(a)).
Since the depot is situated in $d = (1, 1, 0)$, for Merge Algorithm 2 it is sufficient to study the results with left path as *external* and the path on the right as *internal*. We merged the two paths for every possible combination of heights and we can see the results in Table 5.2 with $\alpha = 60°$ and in Table 5.3 with a $\alpha = 85°$.
As we could expect, the Merge Algorithm 2 produces better results over the Merge Algorithm 1. This is because the Merge Algorithm 2 greedily searches the best pair of points on which the merge operation operates, while the first algorithm selects only a pair of points (last point of path 1 and first point of path 2).
With this environment and with this particular division choosed, the Merge Algorithm 2 with both FOVs improves the cost solution of the CPP on the entire map. This rarely happens and it is highly linked to how the map is divided.

Regarding computational time, we can see in bar chart of Figure 5.4 that the larger the FOV, littler the computational time is. This happens because computational time (ct) highly depends on TSP and the number of covering points (when we have a larger FOV, less covering points we have because a wider area is covered on the ground). We can see that both the merge algorithms speed up the ct with respect to the CPP calculated over the entire map. To the advantage of distance cost with the Merge Algorithm 2 that we have seen previously, corresponds a disadvantage in terms of ct. Merge Algorithm 2 spends more time to validate all the possible pairs of points, while Merge Algorithm 1 has only one

	Results with 2-height algorithm				
	cost h_1	cost $h_{1,2}$	cost $h_{1,3}$	cost $h_{1,4}$	cost $h_{1,5}$
	1135.9151	613.9418	479.0015	**426.7281**	469.0775
	Results with Merge Algorithm 1				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	1260.3845	990.5457	943.8797	**931.6059**	960.0786
SingEight2	981.6202	685.7205	649.0638	**579.2133**	659.7604
SingEight3	900.5088	574.4583	520.6913	**505.6983**	555.5524
SingEight4	847.6542	536.3045	482.5376	**467.5446**	507.3987
SingEight5	803.339	521.9893	468.2224	**453.2294**	520.9815
	Results with Merge Algorithm 2				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	1121.8723	911.9299	885.3924	**814.4777**	888.6609
SingEight2	859.5627	629.5705	618.1039	**567.1327**	623.3047
SingEight3	785.1672	512.3199	**464.1151**	503.9135	533.7238
SingEight4	757.8401	482.705	**431.8854**	434.4963	464.695
SingEight5	755.1784	469.246	418.5112	**408.0476**	459.0962

Table 5.2: Comparison of results of Environment A and $\alpha = 60°$

candidate pair of points to merge.

Figure 5.5 shows two paths before the merge operation. Instead, Figure 5.6 shows the same two paths merged into one:

- Figure 5.6(a) shows the path obtained with the Merge Algorithm 1.
- Figures 5.6(b) and 5.6(c) shows the path obtained with the Merge Algorithm 2.

	Results with 2-height algorithm				
	cost h_1	cost $h_{1,2}$	cost $h_{1,3}$	cost $h_{1,4}$	cost $h_{1,5}$
	366.8773	**287.4305**	312.7131	347.265	352.257
	Results with Merge Algorithm 1				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	395.2571	354.5934	**350.8713**	356.5019	434.8559
SingEight2	363.6192	**355.9169**	327.4895	380.6969	376.1821
SingEight3	368.8093	**311.107**	332.6796	379.7168	381.3722
SingEight4	401.4152	**343.7129**	365.2855	410.7659	413.9781
SingEight5	391.3952	**333.6929**	355.2655	404.2225	403.9581
	Results with Merge Algorithm 2				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	**318.5742**	337.5679	346.2264	328.3108	410.746
SingEight2	317.0689	342.8023	419.7275	**312.6508**	371.1372
SingEight3	329.84	**268.3715**	311.0179	281.642	363.5605
SingEight4	336.079	**300.2398**	326.5274	326.3954	391.0104
SingEight5	338.4103	**302.2258**	324.9596	307.2878	388.2201

Table 5.3: Comparison of results of Environment A and $\alpha = 85°$

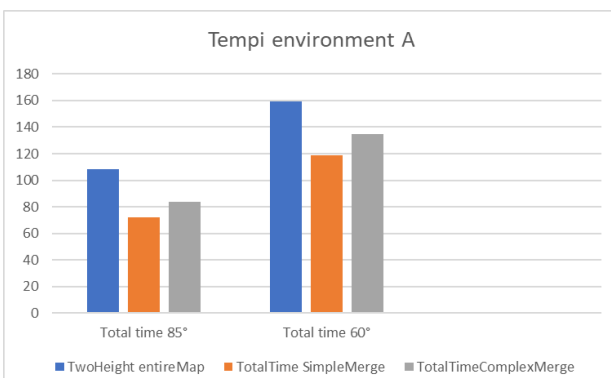
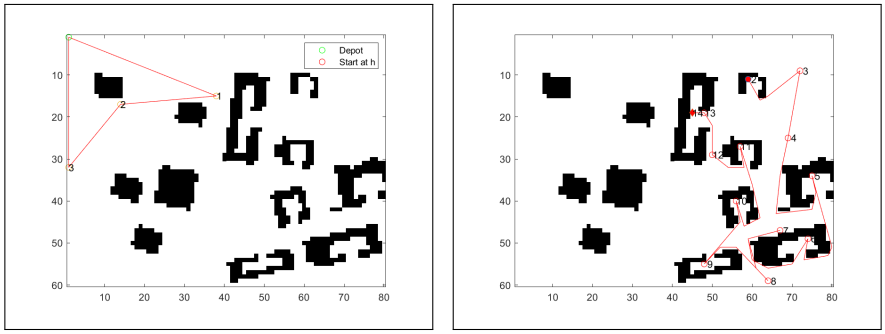


Figure 5.4: Computational time of execution of the algorithm in Environment B with different FOVs



(a) (b)

Figure 5.5: The coverage tours of Environment A with a $\alpha = 85°$. It shows the paths before the merge. (a) Tour at h_2 with SingleHeight algorithm for the type 1 area. (b) Tour at h_2 with the TwoHeight on the type 2 area.

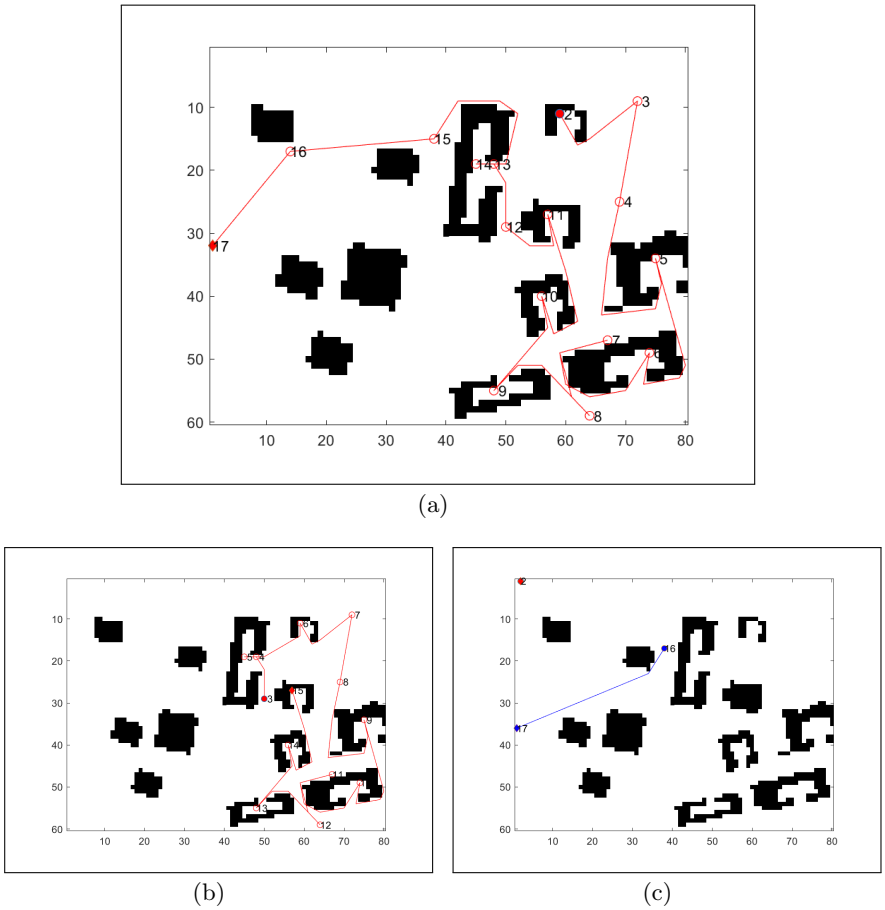


Figure 5.6: The coverage tours of Environment A with a $\alpha = 85°$. (a) Tour at h_2 with Merge Algorithm 1. (b) Tour at h_2 with the Merge Algorithm 2. (c) Tour at h_3 with the Merge Algorithm 2.

5.3.2 Environment B

As we can see in Figure 5.3(b) Environment B is divided in two horizontal sections by a line in the middle such that the upper area is of type 2 and covered by TwoHeight algorithm and the bottom area is covered by a SingleHeight algorithm because is of type 1.
Due to the position of the depot, we have for the Merge Algorithm 2 that the optimal result is always when the upper path is chosen as *external* and the other one as *internal* like we said in Chapter 4.
We merged the two paths for every possible combination of heights and we can see the results in Table 5.4 with $\alpha = 60°$ and in Table 5.5 with $\alpha = 85°$. From these

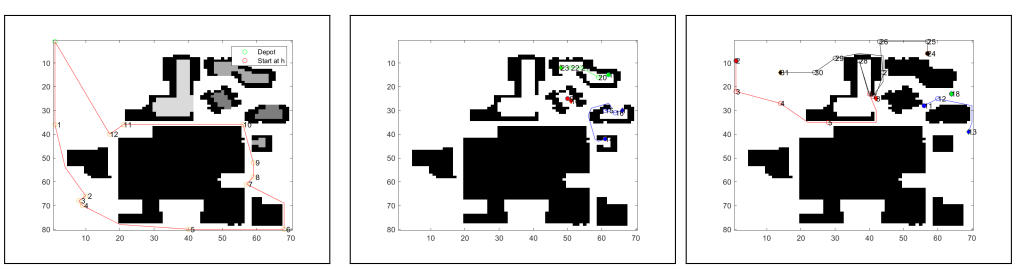
	Results with 2-height algorithm				
	cost h_1	cost $h_{1,2}$	cost $h_{1,3}$	cost $h_{1,4}$	cost $h_{1,5}$
	1082.9853	655.6984	537.7494	**485.6224**	525.3479
	Results with Merge Algorithm 1				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	1153.1592	920.9133	830.7629	**812.5528**	863.7998
SingEight2	1017.2754	727.7656	656.4581	**648.133**	710.5606
SingEight3	944.4761	675.3497	625.4589	**589.71471**	653.5303
SingEight4	909.8902	628.2992	590.8226	**576.4279**	603.8985
SingEight5	919.0005	620.4854	570.7592	**562.2253**	596.807
	Results with Merge Algorithm 2				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	1094.0253	859.2474	816.6287	**809.8505**	841.3824
SingEight2	922.4969	677.6926	647.094	**640.2957**	664.5666
SingEight3	870.6769	620.0147	571.366	**569.4752**	608.9132
SingEight4	822.8105	569.961	520.2932	**488.4063**	562.4777
SingEight5	798.952	585.2586	**482.8592**	502.8436	559.3134

Table 5.4: Comparison of results of Environment B and $\alpha = 60°$

results we can see that in comparison to the CPP over the entire map the results in terms of distance costs are worse. This is because the merge algorithms do not optimize the sequence order of the path points as the TSP does. For example, in Figures 5.8(b) and 5.8(c) we can see a loss of cost distance between point 13 and point 14 because the edge between them intersects the path. For this reason, these results in terms of cost distance can be optimized with an ordering algorithm over the merged path.
Computational time can be seen in bar chart in Figure 5.10.
An example of paths before the merge operation can be seen in Figure 5.7. While paths merged with Merge Algorithm 1 and Merge Algorithm2 can be seen in Figures 5.8 and 5.9 respectively.

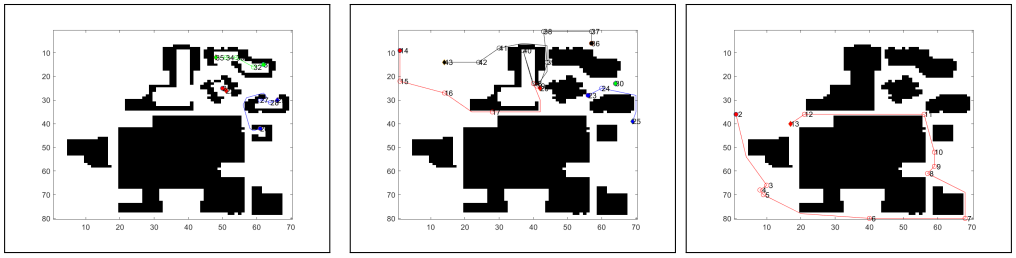
	Results with 2-height algorithm				
	cost h_1	cost $h_{1,2}$	cost $h_{1,3}$	cost $h_{1,4}$	cost $h_{1,5}$
	390.3060	**348.7825**	365.2342	358.2449	363.3426
	Results with Merge Algorithm 1				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	541.2642	536.9533	536.9958	544.1752	**533.8881**
SingEight2	566.1608	553.0682	568.184	576.5365	**537.4695**
SingEight3	587.1449	570.8453	588.126	596.4028	**555.7753**
SingEight4	574.8739	570.3498	570.1987	577.1941	**565.7908**
SingEight5	565.8822	**561.2862**	558.1052	564.7776	563.6747
	Results with Merge Algorithm 2				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	**425.9936**	445.7524	467.9127	473.7134	476.8786
SingEight2	**416.0002**	468.7828	459.9214	466.2329	472.868
SingEight3	**419.5755**	464.7465	464.0838	469.2323	474.76
SingEight4	**430.5171**	474.658	466.2287	477.8185	482.9152
SingEight5	**435.2204**	479.0014	471.5643	479.9337	480.7143

Table 5.5: Comparison of results of Environment B and $\alpha = 85°$



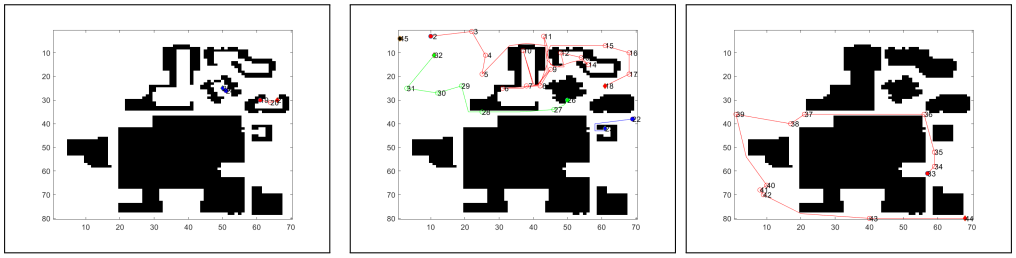
(a) (b) (c)

Figure 5.7: The coverage tours of Environment B with a $\alpha = 60°$. It shows the paths before the merge. (a) Tour at h_5 with SingleHeight algorithm for the type 1 area. (b) Tour at h_1 with the TwoHeight on the type 2 area. (c) Tour at h_4 with the TwoHeight on the type 2 area.



(a) (b) (c)

Figure 5.8: The coverage tour of Environment B with a $\alpha = 60°$ computed with Merge Algorithm 1. The height are in ascending order. (a) Tour at h_1. (b) Tour at h_4. (c) Tour at h_5.



(a) (b) (c)

Figure 5.9: The coverage tour of Environment B with a $\alpha = 60°$ computed with Merge Algorithm 2. The height are in ascending order. (a) Tour at h_1. (b) Tour at h_3. (c) Tour at h_5.

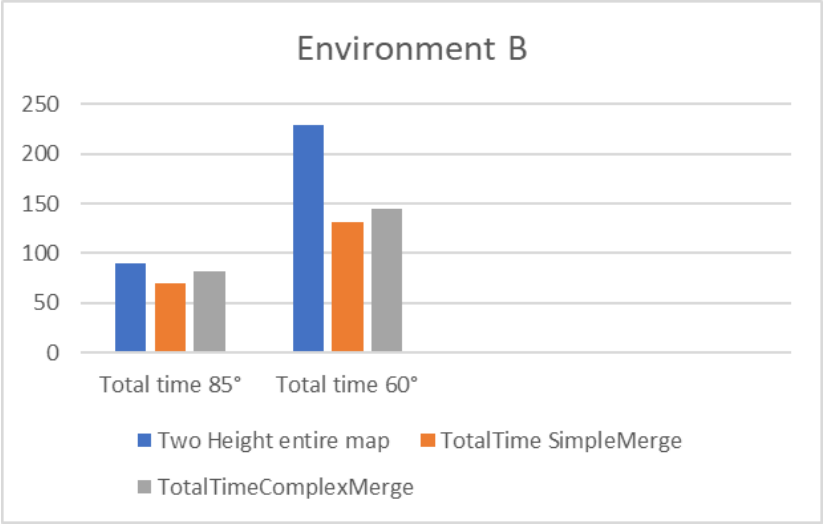


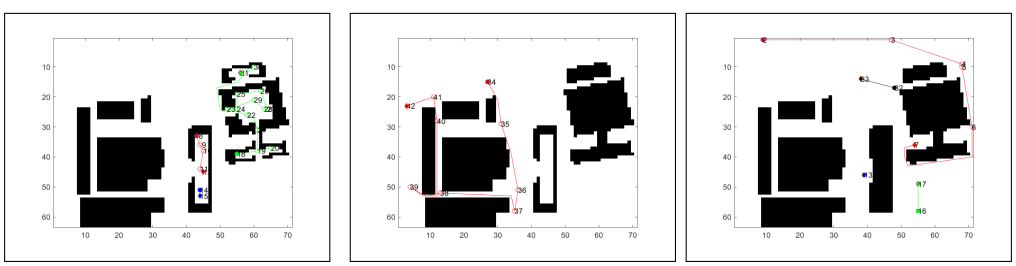
Figure 5.10: Computational time of execution of the algorithm in Environment B with different FOVs

5.3.3 Environment C

From Figure 5.3(c) we clearly see how the division on this environment is done. With respect to the other two environments considered, the distance from the two zones created to the depot is not clearly distinct. For this reason, the optimal solutions with Merge Algorithm 2 must be searched either with path 1 as *external* and path 2 as *internal* or vice versa.
Table 5.6 and Table 5.7 shows the results with $\alpha = 60°$ and $85°$, respectively.
Figures 5.11 and 5.12 show the optimal paths with Merge Algorithm 1 and Merge Algorithm 2 respectively (both FOVs have $\alpha = 60°$).

	Results with 2-height algorithm				
	cost h_1	cost $h_{1,2}$	cost $h_{1,3}$	cost $h_{1,4}$	cost $h_{1,5}$
	993.1080	553.1403	449.9711	440.9308	**423.2185**
	Results with Merge Algorithm 1				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	1018.8887	744.2128	676.3749	674.6393	**658.2405**
SingEight2	905.3447	634.6412	547.1454	561.8577	**545.4589**
SingEight3	897.3258	625.6625	521.7925	542.6041	**520.106**
SingEight4	871.7413	600.078	518.4575	523.2383	**494.5215**
SingEight5	857.3932	585.7299	**495.333**	497.9647	512.7477
	Results with Merge Algorithm 2				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	960.5906	719.5033	647.6455	**637.6901**	647.4389
SingEight2	849.5283	582.5113	542.5369	531.9106	**499.3295**
SingEight3	835.4182	561.5676	465.0597	486.3601	**464.0007**
SingEight4	804.2014	548.7517	**449.6981**	460.5839	455.5957
SingEight5	788.7397	528.3	431.2133	445.8455	**419.1476**

Table 5.6: Comparison of results of Environment C and $\alpha = 60°$

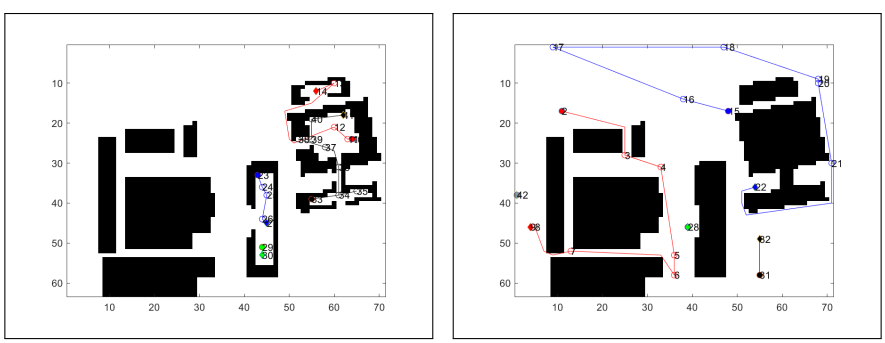


(a) (b) (c)

Figure 5.11: The coverage tour of Environment C with a $\alpha = 60°$ computed with Merge Algorithm 1. The height are in ascending order. (a) Tour at h_1. (b) Tour at h_4. (c) Tour at h_5.

	Results with 2-height algorithm				
	cost h_1	cost $h_{1,2}$	cost $h_{1,3}$	cost $h_{1,4}$	cost $h_{1,5}$
	348.1252	274.0452	**253.5698**	268.2682	276.9552
	Results with Merge Algorithm 1				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	398.2015	**334.2426**	311.9353	334.8074	383.6234
SingEight2	352.619	309.2252	**306.9228**	329.7949	333.0219
SingEight3	380.4107	320.6568	**310.8015**	333.6736	382.1596
SingEight4	386.7044	328.1405	**318.2852**	351.1573	389.6433
SingEight5	390.3236	331.0177	**321.1624**	344.0345	392.5205
	Results with Merge Algorithm 2				
	h_1	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$
SingEight1	333.3615	330.7866	**311.5033**	305.7504	373.3984
SingEight2	306.641	**288.4127**	302.9018	294.4379	332.0501
SingEight3	308.4737	**283.3438**	283.5136	322.5556	330.4208
SingEight4	333.3524	**282.3175**	294.3332	343.7993	358.7993
SingEight5	332.3116	**288.0087**	304.0844	298.2785	336.9355

Table 5.7: Comparison of results of Environment C and $\alpha = 85°$



(a) (b)

Figure 5.12: The coverage tour of Environment C with a $\alpha = 60°$ computed with Merge Algorithm 2. The height are in ascending order. (a) Tour at h_1. (b) Tour at h_5.

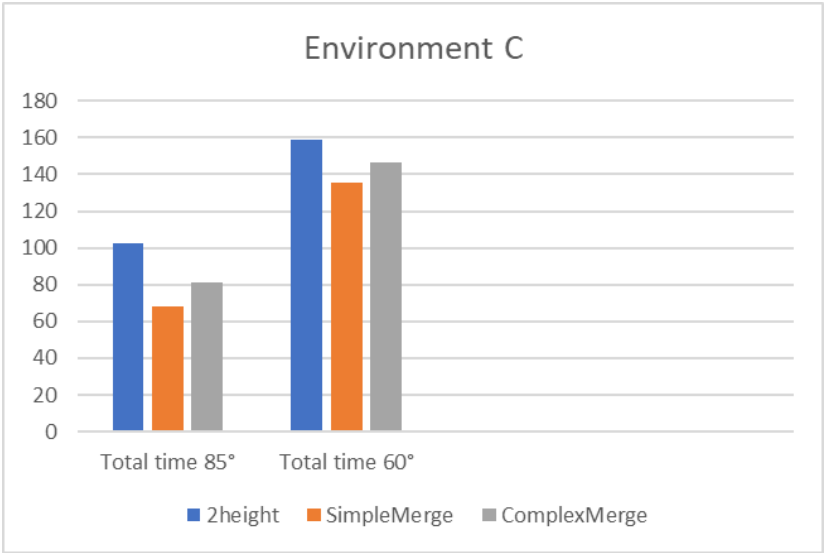


Figure 5.13: Computational time of execution of the algorithm in Environment C with different FOVs

5.3.4 Summary

Tables 5.8 and 5.9 report the optimal results with different Merge Algorithms and FOVs over all the three environments.
From the two tables we can observe that the in general the Merge Algorithm 1 produces worse results in terms of distance cost but the total computational time is less than Merge Algorithm 2. Between the two FOVs used, the larger one produces better results because, as we said in precedence, if we have a very large FOV, from an height h we can cover more cells at h_0 with respect of a small FOV. This means that we will have a quicker computational time due to the fact that the number of covering points is reduced in scenarios with a large FOV.
Regarding the combination of heights that produces the optimal results in the three environments we have different results depending on the FOV used. In scenarios with $\alpha = 60°$ we have both high heights, while with $\alpha = 85°$ we generally have two low heights. In the first case the tour cost decreases with increasing of height. This is due to the fact that going up in height is possible to reduce the number of covering points and so the distances between them (due to the use of Theta*), even if we have an higher cost moving to/from the depot. Instead, with $\alpha = 85°$ the results are different, the tours at intermediate heights are the best. This is due to the fact that having a very large FOV (170°) at a lower height, there is no advantage in going up. Indeed, from a certain height reducing the number of covering points is no longer possible.
Results and heights that produce the optimal result highly depend on the area

to be covered. For example if we apply another division of the environment B we will have different results. If the type 1 zone is bigger we will have a higher height, vice versa if the zone is more little the height will likely reduce. The same will happen with type 2 zone. Appendix A shows the all the heights of the maps used for our experiments.
In Appendix B we can see the optimal coverage paths over the three environments with different FOVs.

Minimum-cost tour with Merge Algorithm 1			
α	Minimum-cost tour	Height	Computational time
		Environment A	
60°	453.2294	$Single_{h_5}, Two_{h_{1,4}}$	119,015 sec
85°	311.107	$Single_{h_3}, Two_{h_{1,2}}$	72,38 sec
		Environment B	
60°	562.2253	$Single_{h_5}, Two_{h_{1,4}}$	130.7115 sec
85°	533.8881	$Single_{h_1}, Two_{h_{1,5}}$	69,752 sec
		Environment C	
60°	306.9228	$Single_{h_2}, Two_{h_{1,3}}$	135,392 sec
85°	318.2447	$Single_{h_2}, Two_{h_{1,2}}$	68,2143 sec

Table 5.8: The minimum-cost coverage tour found by the Merge Algorithm 1 the heights merged and their computational time.

Minimum-cost tour with Merge Algorithm 2			
α	Minimum-cost tour	Height	Computational time
		Environment A	
60°	408.0476	$Single_{h_5}, Two_{h_{1,4}}$	134,8525 sec
85°	268.3715	$Single_{h_3}, Two_{h_{1,2}}$	84 sec
		Environment B	
60°	482.8592	$Single_{h_5}, Two_{h_{1,3}}$	145,3227 sec
85°	416.0002	$Single_{h_1}, Two_{h_{1,1}}$	81,1896 sec
		Environment C	
60°	419.1476	$Single_{h_5}, Two_{h_{1,5}}$	146,6181 sec
85°	320.613	$Single_{h_4}, Two_{h_{1,2}}$	282,3175 sec

Table 5.9: The minimum-cost coverage tour found by the Merge Algorithm 2 the heights merged and their computational time.

5.3.5 TSP solution

A different approach with respect to the merge algorithms is here described.
It is based again on a divide-et-ipera approach. After the environment is divided and the coverage paths for each zone are produced (again, through AGP and TSP), another TSP over all the coverage points is done. This, will produce a path over the entire environment with the points.
The TSP between all the covering points is done in a greedy way over all the heights combination (as we have done with the merge algorithms) We observed that there is a big improvement in the distance cost result and the computational cost is similar to those registered with merge algorithms, at least with the FOV with $\alpha = 85°$. Instead, with the FOV with $\alpha = 60°$ the computational time is very high due to the large number of covering points. As we have seen in precedence, TSP computational time grows linearly with the number of coverage points. So, with this approach when we use a very little FOV sensor, we have a very high computational time.

In Table 5.10 we can see the best results obtained in the three environments with different FOVs. As fo the merge algorithms the results an vary with a different division of the map. We can see that the computational time with $\alpha = 60°$ is very high, also higher to the *cp* of the CPP over the entire map. This, is the most significant drawback of this method because the *cp* will continue to grow reducing the FOV.

Minimum-cost tour with TSP solution			
α	Minimum-cost tour	Height	Computational time
		Environment A	
60°	214.15	$Single_{h_4}, Two_{h_{1,3}}$	133,7003 sec
85°	184.42	$Single_{h_3}, Two_{h_{1,2}}$	69,574 sec
		Environment B	
60°	219	$Single_{h_5}, Two_{h_{1,5}}$	174,3227 sec
85°	118.53	$Single_{h_5}, Two_{h_{1,5}}$	68 sec
		Environment C	
60°	228.54	$Single_{h_5}, Two_{h_{1,4}}$	171,3855 sec
85°	123.04	$Single_{h_3}, Two_{h_{1,2}}$	69,3174 sec

Table 5.10: The minimum-cost coverage tour found by the Merge Algorithm 2 the heights merged and their computational time.

Chapter 6

Conclusions and future works

In this work, we implemented a divide-et-impera approach to solve multiple 2D CPP (Coverage Path Planning) problems over the 2D grids in which a 3D environment is decomposed with the aim to decrease the computational cost of a CPP problem solution. In our case the CPP produces a tour that allows an UAV (Unmanned Aerial Vehicle) with a given covering sensor to cover a target area at the ground.

Our criterion to divide the initial problem is based on the type of environment. We have two different types of possible environments.

Type 1 considers the environments with a monotone behavior of the covering function. This means that the area covered at given height is always included in the area covered at a higher height.
Type 2 includes the environments in which the area that can be covered at a given height is a subset of the area covered at any lower heights.
Each zone in which the initial map is decomposed, is covered separately through the AGP (Art Gallery Problem) to find the minimum number of covering points and the TSP (Travel Salesman Problem) to connect them, using Theta* to calculate the distance cost.

After all the paths are calculated, a merge algorithm is required to produce a solution to the initial problem. We proposed two different merge algorithms. One produces high distance cost paths with less computational time, while the second gives better results in terms of distance cost but with much more computational time.

We implemented and tested our algorithms in three different environments using different FOVs (Field Of View) for the robot's sensor and discretizing each 3D environment over six heights $H = h_0, h_1, h_2, h_3, h_4, h_5$.

Results highly depend on the environments and on how the division of the map is done. With a large FOV we have better results with lower heights, vice versa with a smaller FOV we have that the solution is produced by a combination of high heights.
We saw the differences between the two merge algorithms in terms of distance cost and computational time. The former is better with Merge Algorithm 2, the latter with Merge Algorithm 1.

Future works could investigate more deeply the aspect of the map division in order to find the particular division that produces better results. Another future work, could be a study to further improve the computational time, discarding some heights that surely don't give a better solution with respect to other heights.

Appendix A

Occupancy grid maps in all the three environments

Here we can see how the three environments used for our experiments develop vertically.

- Figure A.1 shows the occupancy grid maps of Environment A
- Figure A.2 shows the occupancy grid maps of Environment B
- Figure A.3 shows the occupancy grid maps of Environment C

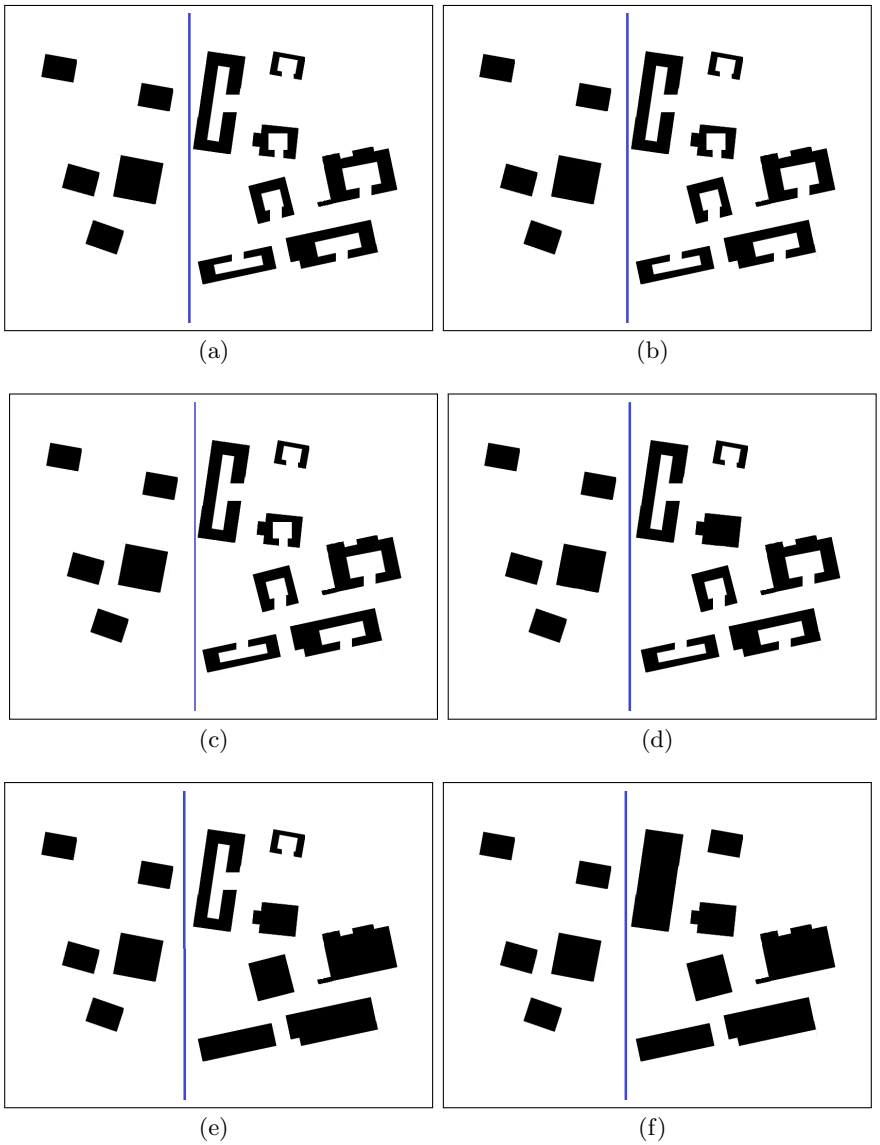


Figure A.1: Six grids represent the occupancy grid maps at different heights of Environment A in ascending order starting from h_0 (the ground) to h_5. Also the division is showed.

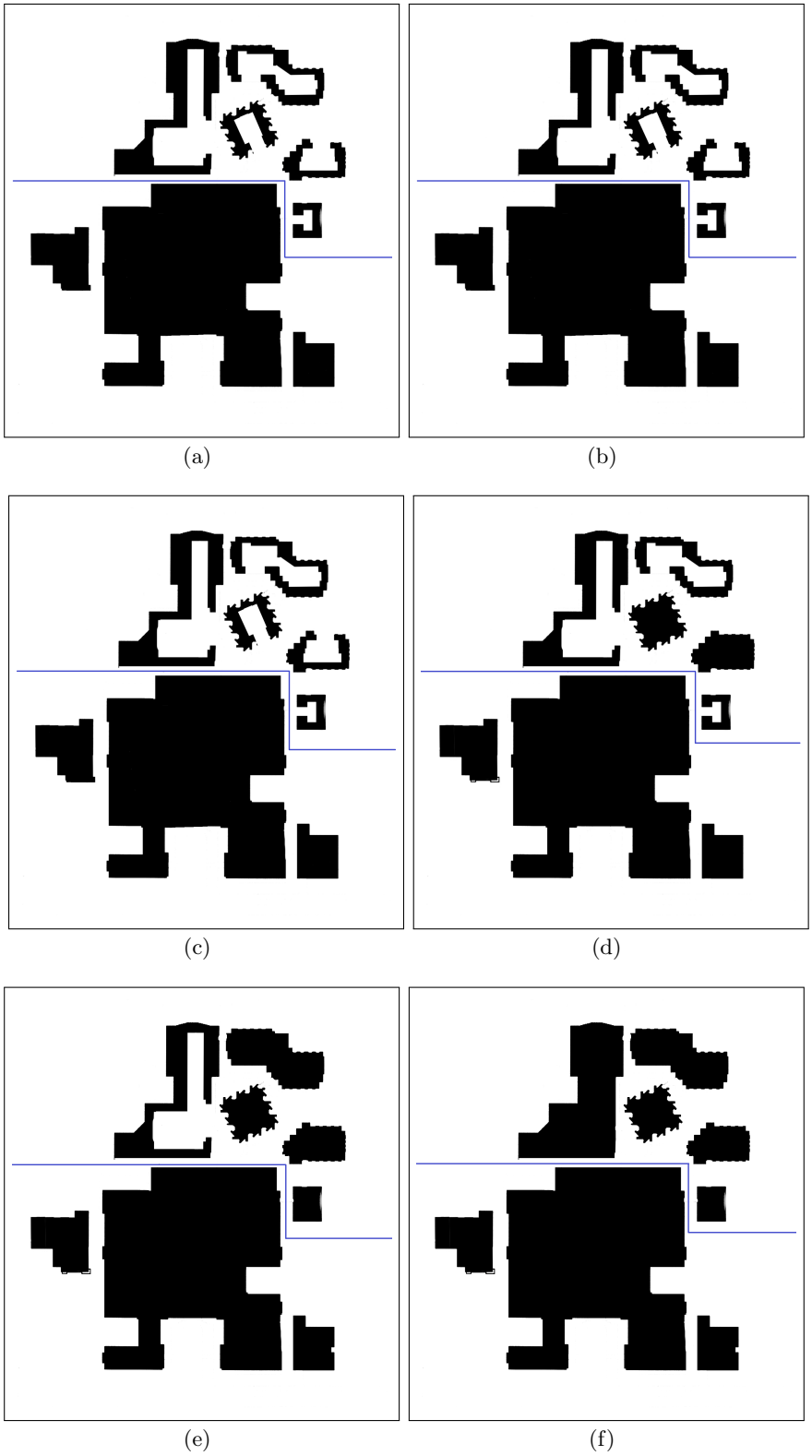


Figure A.2: Six grids represent the occupancy grid maps at different heights of Environment B in ascending order starting from h_0 (the ground) to h_5. Also the division is showed.

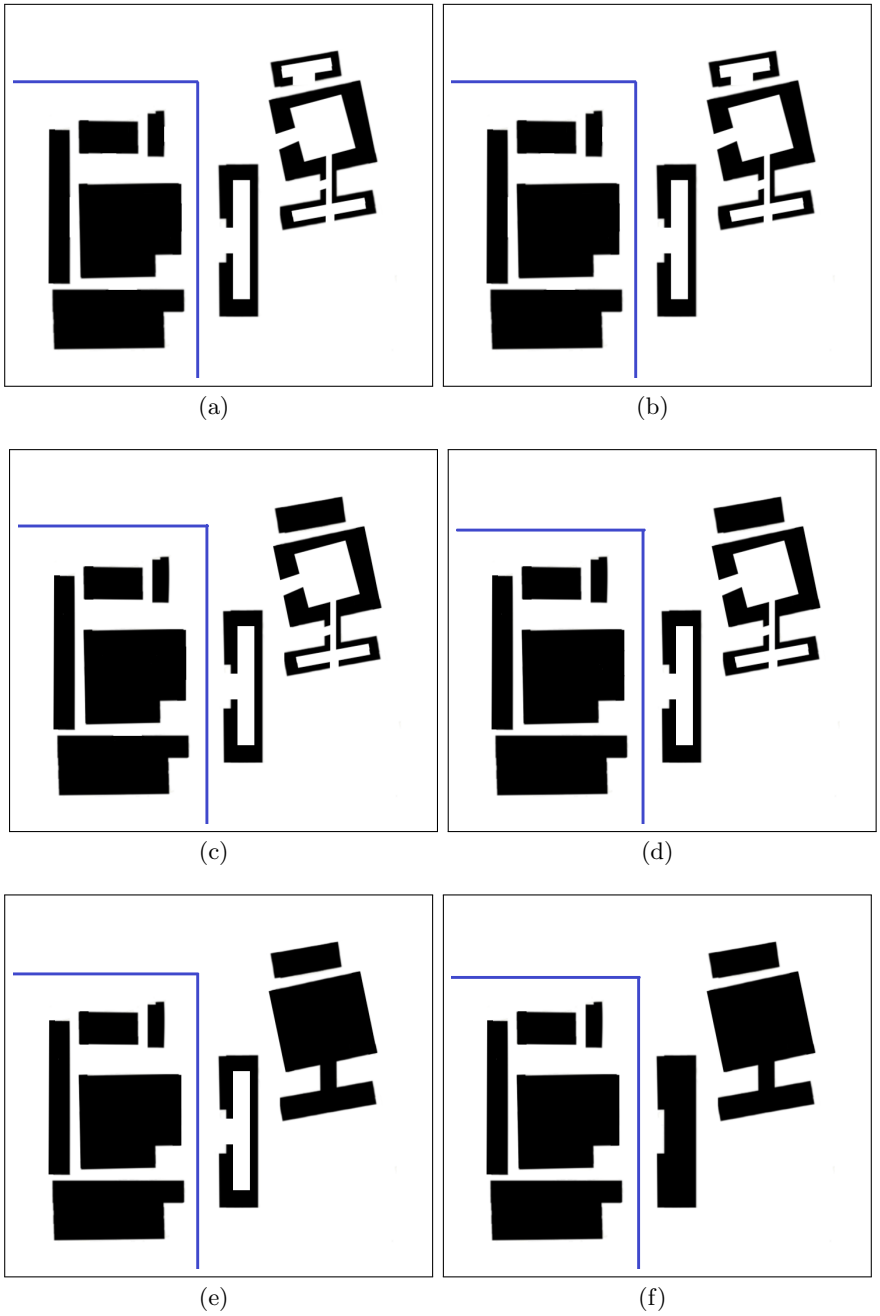


Figure A.3: Six grids represent the occupancy grid maps at different heights of Environment C in ascending order starting from h_0 (the ground) to h_5. Also the division is showed.

Appendix B

Optimal paths maps for all the environments and different FOVs

Here, is presented a list of figures that represent the optimal paths with both merge algorithms for each environment.

Environment A

The following images are:

- Figures B.1(a), B.1(b) and B.1(c) show the optimal path with $\alpha = 60°$ and Merge Algorithm 1.
- Figures B.1(d), B.1(e) and B.1(f) show the optimal path with $\alpha = 60°$ and Merge Algorithm 2.
- Figure B.2(a) shows the optimal path with $\alpha = 85°$ and Merge Algorithm 1.
- Figure B.2(b) and B.2(c) show the optimal path with $\alpha = 85°$ and Merge Algorithm 2.

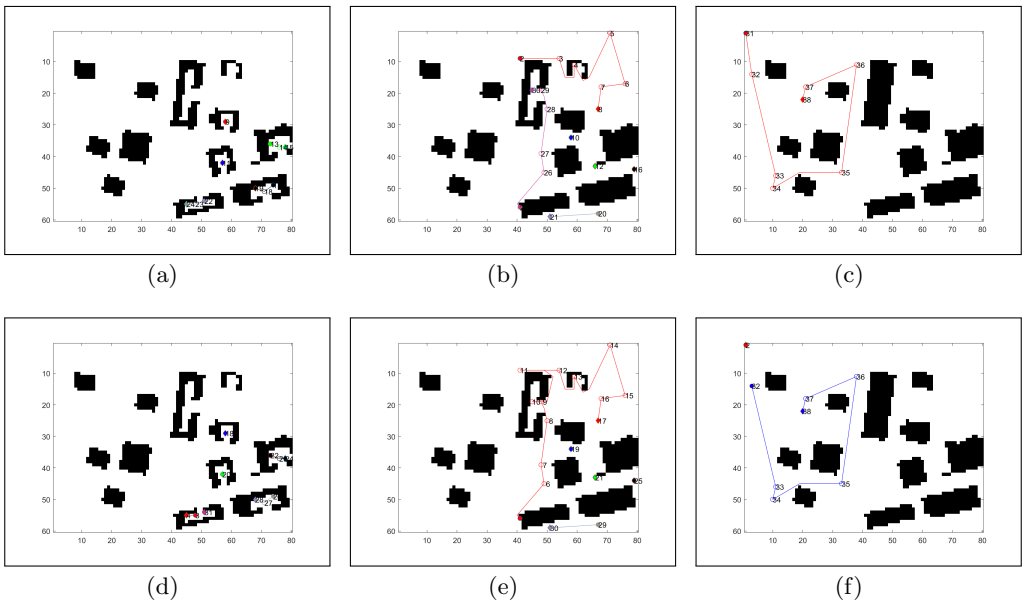


Figure B.1: The coverage tours of Environment A with a $\alpha = 60°$. (a) Tour at h_1 with Merge Algorithm 1. (b) Tour at h_4 with the Merge Algorithm 1. (c) Tour at h_5 with the Merge Algorithm 1. (d) Tour at h_1 with Merge Algorithm 2. (e) Tour at h_4 with the Merge Algorithm 2. (f) Tour at h_5 with the Merge Algorithm 2.

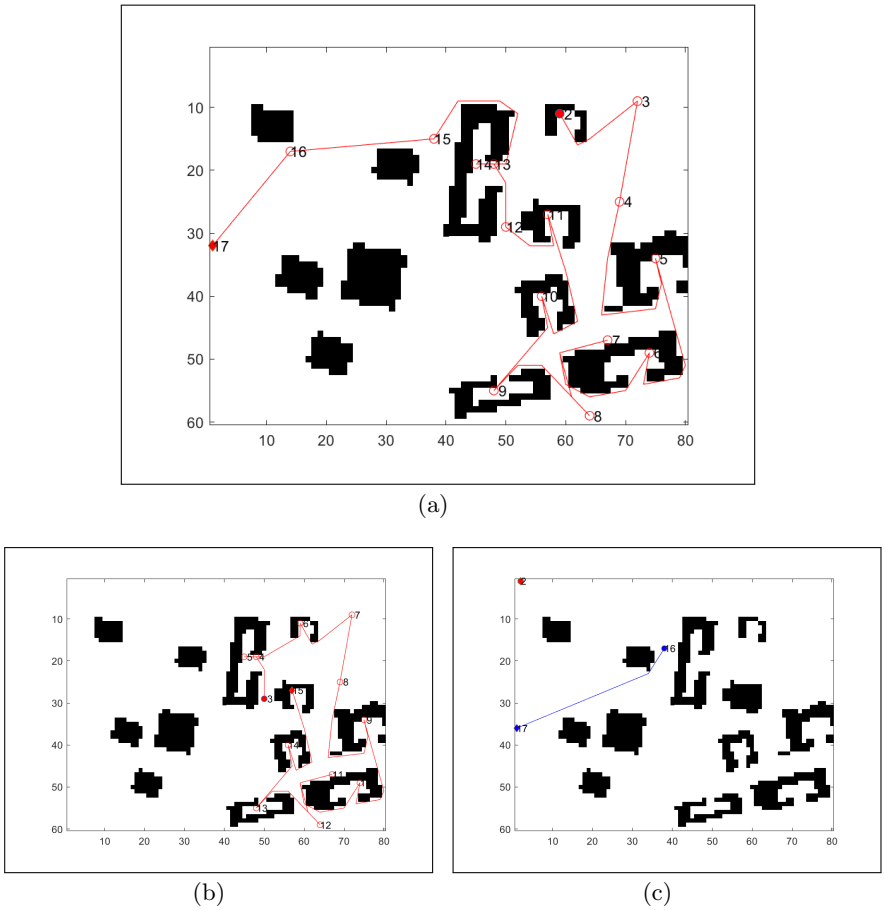


Figure B.2: The coverage tours of Environment A with a $\alpha = 85°$. (a) Tour at h_2 with Merge Algorithm 1. (b) Tour at h_2 with the Merge Algorithm 2. (c) Tour at h_3 with the Merge Algorithm 2.

Environment B

The following images are:

- Figures B.3(a), B.3(b) and B.3(c) show the optimal path with $\alpha = 60°$ and Merge Algorithm 1.
- Figures B.3(d), B.3(e) and B.3(f) show the optimal path with $\alpha = 60°$ and Merge Algorithm 2.
- Figure B.4(a) and B.4(a) shows the optimal path with $\alpha = 85°$ and Merge Algorithm 1.

- Figure B.4(c) and B.4(d) show the optimal path with $\alpha = 85°$ and Merge Algorithm 2.

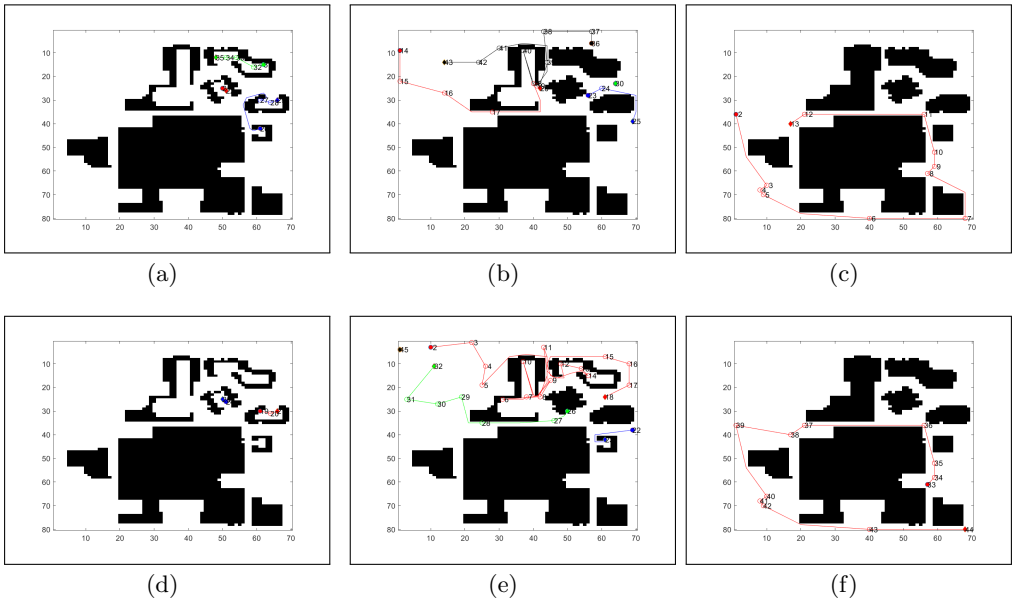
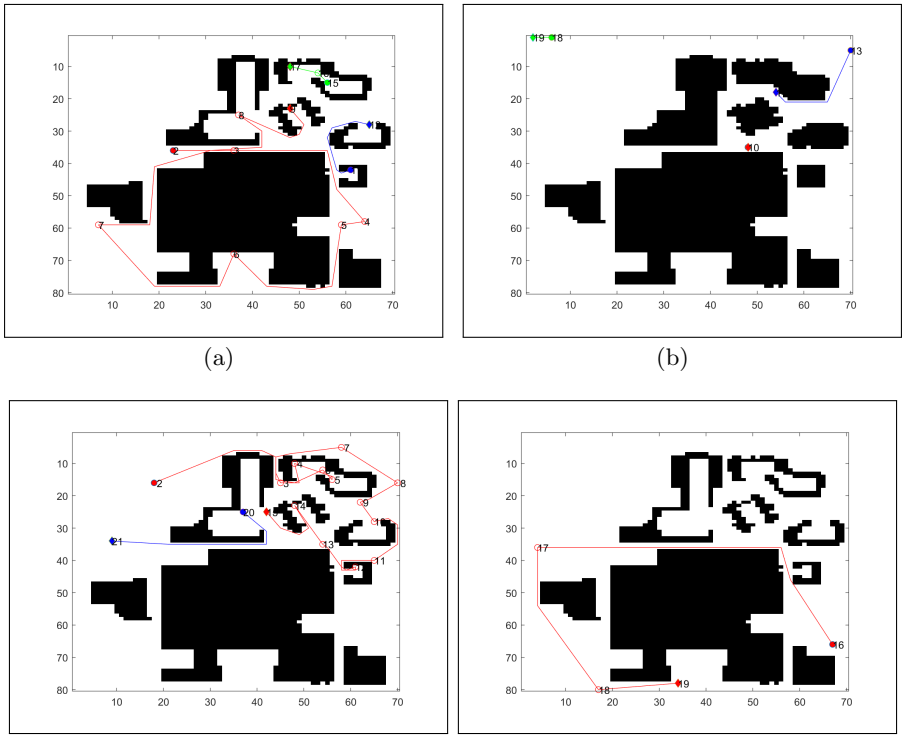


Figure B.3: The coverage tour of Environment B with a $\alpha = 60°$ computed. (a) Tour at h_1 with Merge Algorithm 1. (b) Tour at h_4 with Merge Algorithm 1. (c) Tour at h_5 with Merge Algorithm 1. (d) Tour at h_1 with Merge Algorithm 2. (e) Tour at h_3 with Merge Algorithm 2. (f) Tour at h_5 with Merge Algorithm 2.



(a) (b) (c) (d)

Figure B.4: The coverage tour of Environment B with a $\alpha = 85°$ computed. (a) Tour at h_1 with Merge Algorithm 1. (b) Tour at h_5 with Merge Algorithm 1. (c) Tour at h_1 with Merge Algorithm 2. (d) Tour at h_2 with Merge Algorithm 2.

Environment C

The following images are:

- Figures B.5(a), B.5(b) and B.5(c) show the optimal path with $\alpha = 60°$ and Merge Algorithm 1.
- Figures B.5(d) and B.5(d) show the optimal path with $\alpha = 60°$ and Merge Algorithm 2.
- Figure B.6(a), B.6(b) and B.6(c)shows the optimal path with $\alpha = 85°$ and Merge Algorithm 1.
- Figure B.6(d),B.6(e) and B.6(f) show the optimal path with $\alpha = 85°$ and Merge Algorithm 2.

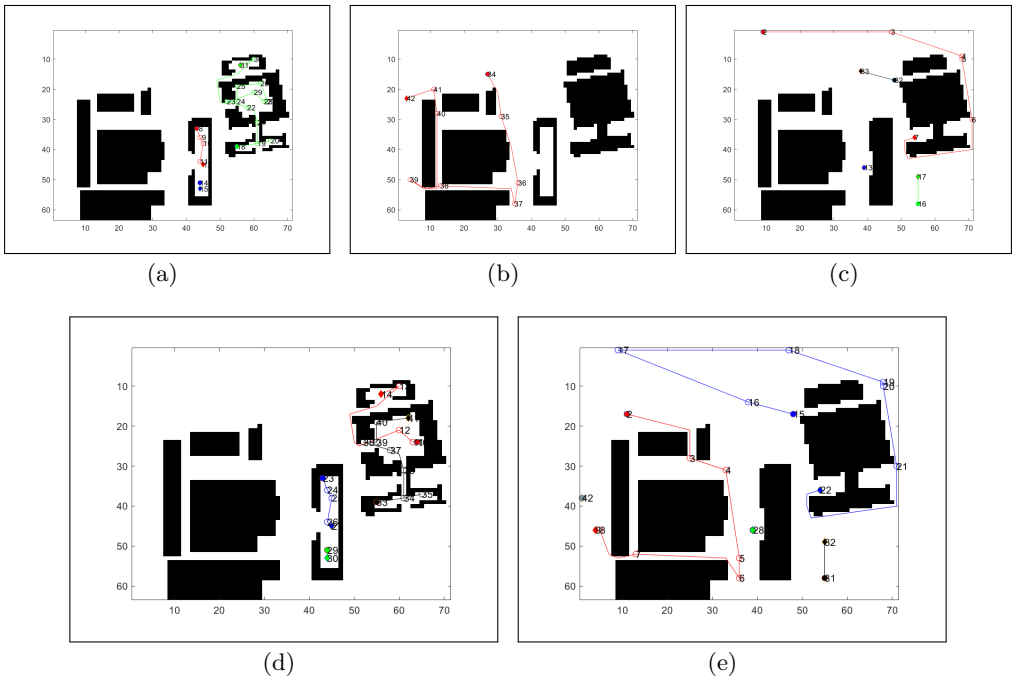


Figure B.5: The coverage tour of Environment C with a $\alpha = 60^{\circ}$ computed.(a) Tour at h_1 with Merge Algorithm 1. (b) Tour at h_4 with Merge Algorithm 1. (c) Tour at h_5 with Merge Algorithm 1. (d) Tour at h_1 with Merge Algorithm 2. (e) Tour at h_5 with Merge Algorithm 2.

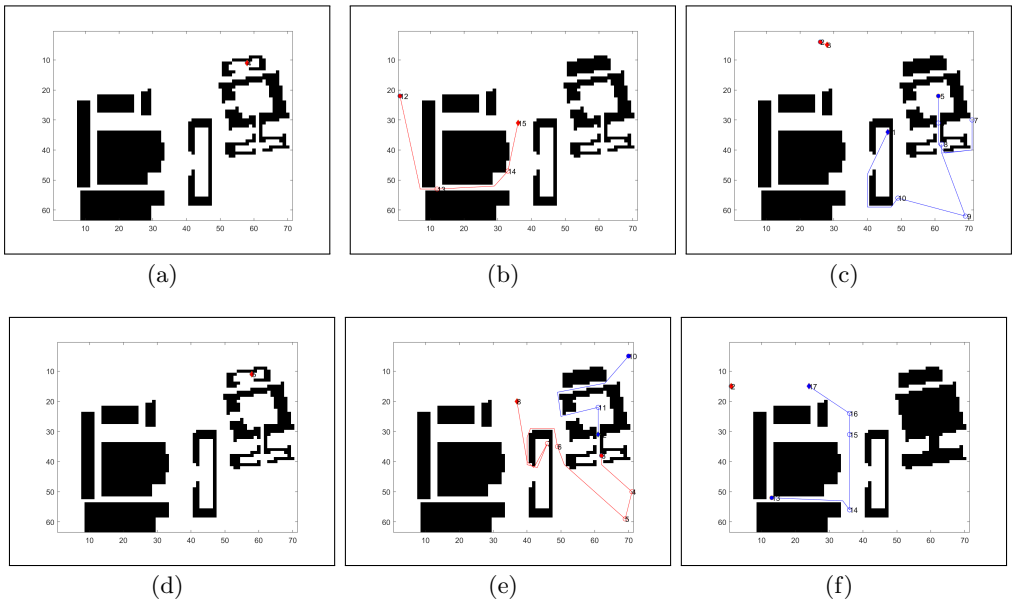


Figure B.6: The coverage tour of Environment C with a $\alpha = 85°$ computed.(a) Tour at h_1 with Merge Algorithm 1. (b) Tour at h_2 with Merge Algorithm 1. (c) Tour at h_3 with Merge Algorithm 1. (d) Tour at h_1 with Merge Algorithm 2. (e) Tour at h_2 with Merge Algorithm 2. (f) Tour at h_4 with Merge Algorithm 2.

Bibliography

[1] Ercan U Acar, Howie Choset, Alfred A Rizzi, Prasad N Atkar, and Douglas Hull. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21(4):331–344, 2002.

[2] Francesco Amigoni and Alessandro Riva. Path planning for ground covering with an uav moving at discrete heights. In *International Joint Conference on Artificial Intelligence – European Conference on Artificial Intelligence / International Conference on Machine Learning / International Conference on Autonomous Agents and Multiagent Systems*. Federated AI for Robotics Workshop (FAIR), 2018.

[3] Muhammad Asif Arain, Marco Trincavelli, Marcello Cirillo, Erik Schaffernicht, and Achim J Lilienthal. Global coverage measurement planning strategies for mobile robots equipped with a remote gas sensor. *Sensors*, 15(3):6845–6871, 2015.

[4] Antonio Barrientos, Julian Colorado, Jaime del Cerro, Alexander Martinez, Claudio Rossi, David Sanz, and João 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*, 28(5):667–689, 2011.

[5] Nicola Basilico and Stefano Carpin. Deploying teams of heterogeneous uavs in cooperative two-level surveillance missions. In *Proceedings of the 2015 IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 610–615. IEEE, 2015.

[6] Andreas Bircher, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. In *Proocedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6423–6430. IEEE, 2015.

[7] Bovisa campus Politecnico di Milano. Environment B. https://goo.gl/ztFzUj, 2018.

[8] John F Canny and Ming C Lin. An opportunistic global path planner. *Algorithmica*, 10(2-4):102–120, 1993.

[9] Peng Cheng, James Keller, and Vijay Kumar. Time-optimal uav trajectory planning for 3d urban structure coverage. In *Proceedings of the 2008 IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 2750–2757. IEEE, 2008.

[10] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon decomposition. In *Proceedings of the 1997 International Conference on Field and Service Robotics*, pages 3–91, 1997.

[11] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[12] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[13] Tim Danner and Lydia E Kavraki. Randomized planning for short inspection paths. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 971–976. IEEE, 2000.

[14] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale travelling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[15] Brendan Englot and Franz Hover. Planning complex inspection tasks using redundant roadmaps. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 232–240. ACM, 2011.

[16] Brendan Englot and Franz S Hover. Sampling-based coverage path planning for inspection of complex structures. In *Proceedings of the 22nd International Conference Automated Planning and Scheduling*, pages 29–37, 2012.

[17] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[18] Yoav Gabriely and Elon Rimon. Spiral-stc: An on-line coverage algorithm of grid environments by a mobile robot. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 954–960. IEEE, 2002.

[19] Enric Galceran, Ricard Campos, Narcıs Palomeras, David Ribas, Marc Carreras, and Pere Ridao. Coverage path planning with real-time replanning and surface reconstruction for inspection of three-dimensional underwater structures using autonomous underwater vehicles. *Journal of Field Robotics*, 32(7):952–983, 2015.

[20] Enric Galceran and Marc Carreras. Efficient seabed coverage path planning for asvs and auvs. In *Proceedings of the 2012 IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 88–93. IEEE, 2012.

[21] Greta Ghiotti. Planning paths for covering environments with uavs moving at discrete heights. Master's thesis, Politecnico di Milano, Scuola di Ingegneria Industriale e dell'Informazione – Dipartimento di Elettronica, Informazione e Bioingegneria, 2018.

[22] Héctor González-Banos. A randomized art-gallery algorithm for sensor placement. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 232–240. ACM, 2001.

[23] Héctor González-Banos and Jean-Claude Latombe. Planning robot motions for range-image acquisition and automatic 3d model construction. *AAAI Fall Symposium*, 1998.

[24] Gurobi. Gurobi reference manual. https://goo.gl/zRxUfv, 2018.

[25] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[26] Susan Hert, Sanjay Tiwari, and Vladimir Lumelsky. A terrain-covering algorithm for an auv. In Junku Yuh, Tamaki Ura, and George A Bekey, editors, *Underwater Robots*, pages 17–45. Springer, 1996.

[27] Ray Jarvis. Distance transform based path planning for robot navigation. In Yuan-Fang Zheng, editor, *Recent Trends in Mobile Robots*, pages 3–31. World Scientific, 1993.

[28] D Lee and Arthurk Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[29] Tae-Seok Lee, Jeong-Sik Choi, Jeong-Hee Lee, and Beom-Hee Lee. 3-d terrain covering and map building algorithm for an auv. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4420–4425. IEEE, 2009.

[30] Ivan Maza and Anibal Ollero. Multiple uav cooperative searching operation using polygon area decomposition and efficient coverage algorithms. In Rachid Alami, Raja Chatila, and Hajime Asama, editors, *Distributed Autonomous Robotic Systems 6*, pages 221–230. Springer, 2007.

[31] L H Nam, Loulin Huang, X J Li, and Jianfeng Xu. An approach for coverage path planning for uavs. In *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, pages 411–416. IEEE, 2016.

[32] Mccormick Northwestern. Environment B. https://goo.gl/XvcBcq, 2018.

[33] Patrick A Plonski and Volkan Isler. Approximation algorithms for tours of height-varying view cones. *The International Journal of Robotics Research*, 38(2–3):224–235, 2019.

[34] Ioannis Rekleitis, Ai Peng New, Edward Samuel Rankin, and Howie Choset. Efficient boustrophedon multi-robot coverage: an algorithmic approach. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):109–142, 2008.

[35] Alessandro Riva and Francesco Amigoni. A grasp metaheuristic for the coverage of grid environments with limited-footprint tools. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 484–491. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[36] Seyed Abbas Sadat, Jens Wawerla, and Richard VAughan. Fractal trajectories for online non-uniform aerial coverage. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2971–2976. IEEE, 2015.

[37] McKenna Mout Site. Environment A. https://goo.gl/T1bqH4, 2018.

[38] Petr Slavık. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–254, 1997.

[39] Yoichi Tomioka, Atsushi Takara, and Hitoshi Kitazawa. Generation of an optimum patrol course for mobile surveillance camera. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(2):216–224, 2012.

[40] João Valente, Jaime Del Cerro, Antonio Barrientos, and David Sanz. Aerial coverage optimization in precision agriculture management: A musical harmony inspired approach. *Computers and electronics in agriculture*, 99:153–159, 2013.

[41] João Valente, David Sanz, Jaime Del Cerro, Antonio Barrientos, and Miguel Ángel de Frutos. Near-optimal coverage trajectories for image mosaicing using a mini quad-rotor over irregular-shaped fields. *Precision agriculture*, 14(1):115–132, 2013.

[42] Anqi Xu, Chatavut Viriyasuthee, and Ioannis Rekleitis. Optimal complete terrain coverage using an unmanned aerial vehicle. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2513–2519. IEEE, 2011.

[43] Simon X Yang and Chaomin Luo. A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):718–724, 2004.

[44] Alexander Zelinsky, Ray A Jarvis, JC Byrne, and Shinichi Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of International Conference on Advanced Robotics*, volume 13, pages 533–538, 1993.