Warsaw University of Technology
Faculty of Power and Aeronautical Engineering
Division of Theory of Machines and Robots

Diploma Thesis
Master Degree

Francesco Wanderlingh

Swarm of autonomous MAVs in urban Search and Rescue applications

Student ID: 263870
Robotics and Automation
European Master on Advanced Robotics (EMARO)

Supervisor: Prof. Wojciech Szynkiewicz - WUT Poland
Co-Supervisor: Prof. Antonio Sgorbissa - UNIGE Italy
Co-Supervisor: Prof. Cristiano Nattero - UNIGE Italy

Warsaw, June 2014

Statement of the Author of the Thesis
(Oświadczenie autora pracy)

Being aware of my legal responsibility, I certify that this diploma:
(Świadom odpowiedzialności prawnej oświadczam, że przedstawiona praca dyplomowa:)

- has been written by me alone and does not contain any content obtained in a manner inconsistent with the applicable rules,
(została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obwiązującymi przepisami,)

- has not been previously subject to the procedures for obtaining professional title or degree at a university.
(nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego lub stopnia naukowego w wyższej uczelni.)

Furthermore I declare that this version of the diploma thesis is identical with the electronic version attached.
(Oświadczam ponadto, że niniejsza wersja pracy jest identycna z załączoną wersją elektroniczną.)

.....	.....
date	author's signature
data	*podpis autora (autorów) pracy*

Keywords: Complete Terrain Coverage, Autonomous Robots, Unmanned Aerial Vehicles, Search and Rescue

Słowa kluczowe: całkowite pokrycie terenu, roboty autonomiczne, bezzałogowe pojazdy latające, poszukiwanie i ratunek

Swarm of autonomous MAVs in Urban Search and Rescue applications

by
Francesco Wanderlingh

Abstract

In the framework of swarm robotics involved in search and rescue applications, the main purpose of this thesis is to investigate complete terrain coverage algorithms. These kinds of algorithms are intended in the first place to perform a fast monitoring by means of aerial imaging, but many other applications are possible. Two main classes can be identified in the algorithms developed for the terrain coverage: online and offline coverage methods. For the online class the algorithms taken into account are Node Counting, Learning Real-Time A*, Edge Counting and PatrolGRAPH*. For the offline class instead the coverage problem was formulated as a Vehicle Routing Problem and solved with a greedy approach. In a first phase the algorithms were tested in a 3D simulation environment and a large number of results was collected and compared. Finally the whole framework was tested performing the coverage using real multi-copters.

Zastosowania roju autonomicznych miniaturowych statków powietrznych w poszukiwaniu i ratownictwie miejskim

przez
Francesco Wanderlingh

Streszczenie

Głównym celem pracy jest analiza algorytmów całkowitego pokrycia terenu z użyciem roju miniaturowych statków powietrznych w poszukiwaniu i ratownictwie miejskim. Algorytmy te są przeznaczone, przede wszystkim, do szybkiego monitorowania terenu z użyciem obrazowania z powietrza, aczkolwiek możliwych jest wiele innych zastosowań. Wyróżnia się dwie główne klasy algorytmów przeszukania terenu: metody *on-line* i *off-line*. Do pierwszej z nich można zaliczyć algorytmy: Node Counting, Learning Real-Time A*, Edge Counting and PatrolGRAPH*. W podejściu *off-line* zadanie pokrycia terenu zostało sformułowane jako problem marszrutyzacji i rozwiązano go za pomocą algorytmów zachłannych przeszukiwania grafu. W pierwszej fazie badań algorytmy były testowane w symulacji w stworzonym środowisku 3D. Wykonano wiele testów, a ich wyniki zostały zebrane i porównane. Następnie przeprowadzono eksperymenty pokrycia terenu z użyciem rzeczywistych multikopterów.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General motivation and objectives

When natural disaster occur, a prompt intervention is a key factor. In order to organize effective rescue missions, it is fundamental to quickly gather information about collapsed buildings, damaged roads and, above all, victims. To this extent, autonomous Micro Aerial Vehicles (MAVs) are a promising technology to survey the stricken area.

The introduction of unmanned aerial vehicles (UAVs) has revolutionized military operations all over the world. In the past year, the U.S. reached an important milestone in that Air Force now has more UAVs than manned aircraft, while Israel and the United Kingdom had recent significant advances in UAV heavy-lift capacity [10]. These advances are not only limited to the military as the international civil sector also looks to such unmanned technologies to aid operations such as fighting forest fires, undersea exploration, monitoring wildlife, inspecting bridges, and supporting first responders such as police and rescue organizations. UAV expenditures alone are predicted to more than double in the next ten years, and are expected to exceed $80 billion [10]. So far this century there have been more than 1000 fatal earthquakes causing a total loss of life exceeding 1.5 million people. Reducing the loss of life is the primary priority of most earthquake protection strategies, and yet the processes that contribute to death tolls and the best strategies for reducing injury levels are not well understood. According to [9], structural collapses are responsible for 75% of deaths in earthquakes. The factors influencing the number of victims per building collapse fall into five major categories M1 to M5. Being M3 the "occupants trapped by collapse".

In all three cases it's evident how a flying vehicle can be useful to reach the areas struck by the earthquake. When disasters such as this occur, a prompt inter-

vention is a key factor: the probability of saving lives decreases dramatically after the initial hours. In order to organize effective rescue missions, it is fundamental to quickly gather information about collapsed buildings, damaged roads and, above all, victims. To this extent, autonomous Micro Aerial Vehicles (MAV) are a promising technology to survey the stricken area. These applications usually require complete autonomy in navigation, small size of the air vehicle to access also confined out and indoor spaces and low power consumption for extended operation times.

In the framework of swarm robotics involved in search and rescue (SAR or S&R) applications, the main purpose of this thesis research is to investigate complete coverage algorithms. These kind of algorithms are intended in the first place to perform a fast monitoring by means of aerial imaging, but many other applications are possible.

The branch of swarm robotics is relatively new, and the application of swarm robotics to flying vehicles is even more innovative. The capacity of having multiple robots with such mobility in fact opens new possibilities with respect to terrain exploration.

I developed in parallel two different approaches: real-time and optimised coverage. They fit different hardware and swarm-type requirements, but they share the same mathematical base, graph theory.

To test the algorithms a 3D simulation environment was created and the simulated robots are controlled using the ROS robotic platform. This was mandatory choice for testing online algorithms, but nevertheless a good way to visualise the results for pre-processed optimised coverage. The simulated environment has been represented as a graph using a convention shown in the next chapter, on which the algorithms operate to perform the coverage. As a last step the algorithms were also tested with success on a pair of real multi-copters available in the laboratory, proving that a consistent framework is available for immediate application.

1.2 The Coverage Problem

This thesis work deals with the problem of finding the most effective algorithm to solve the coverage problem. Let's then start defining what the coverage problem deals with and how we can solve it.

The are two major classes in which we can divide coverage problems: *static* coverage and *dynamic* coverage. Given a finite area with obstacles to cover, the static coverage solves the problem of finding the best robot positions such that visibility of the area is maximized. In this case then there will be some which will

not be monitored by the robots (blind spots) and for our SAR purposes this is unacceptable. For this reason in the thesis we opted for a dynamic approach to coverage. In a dynamic coverage the robot will have to cover the area by exploring it thoroughly, so even if at each time step we just observe a little portion of the terrain, the final knowledge of the area is complete. If for example we have to find a victim in a stricken environment and the person is hidden in a blind spot, the static coverage does not ensures that the victim will be detected, while the dynamic coverage does. A static coverage moreover does not observe the area uniformly; the regions near to the robot will be well monitored while far one will be observed only marginally.

Now in the the field of dynamic coverage we can identify two classes:

- **Offline coverage**: the paths are pre-calculated with a routing algorithm, before the robot start exploring the environment
- **Online coverage**: the robots decide their path while they are exploring the environment, by making decisions based on the current knowledge they have on the coverage completion

An important aspect to take into account is whether the coverage is meant to be performed by a single robot, or by a group of robots. The main difference between the two is that in the single robot the robot does not have to interact with other moving units and this simplifies the planning and does not require a inter-robot communication system. In the multi-robot coverage instead the communication between units is essential to take advantage of a collaborative behaviour. In this thesis the main objective has been to build a flexible software framework that can be easily extend from a single-robot to multi-robot configuration seamlessly.

For known environments it is important to accomplish the coverage task in the minimum possible time and using this a priori knowledge we can construct an algorithm that computes the optimal paths to completely cover the area. This is possible decomposing the environment in a finite number of spots to visit (see next section, 1.3.2) and by modelling the coverage as a routing problem where we have to visit all the location at least once. But using offline algorithms to find optimal path turns out to be not so trivial, optimal solution are really hard to compute, actually NP-hard [26], so in practice heuristic and deterministic methods have been developed that find acceptably good solutions for the VRP. This involves a discrete computation capability and in the perspective of using the algorithms to guide autonomous robots this can represent a limitation. That is why several online strategies have also been investigated. From a computational point of view

online algorithms are far less challenging. In an online coverage basically the robot has the only simple task of choosing what step to take next, basing is decision of the current knowledge it has on the current level of completion of the coverage. It also has the advantage of being more flexible; think of a situation where a robot fails due to a crash: in an offline coverage the path are pre-calculated so the path assigned to that robot will not be covered, in an online case instead the robots will continue exploring the area till they know that the whole area is covered. They also fit better the swarm robotics approach where complex behaviours emerges from the sum of many simple ones.

As a final remark since performing SLAM (simultaneous localization and mapping) is out of the scope of this thesis the terrain is assumed to be known. Having in mind that by using any modern web mapping service the topology of the area can be retrieved this can be a reasonable assumption in real world applications.

The path planning required to perform the complete coverage task must be build on top of an abstract representation of the terrain. This abstract representation can be obtained through one of the various methods discussed in the next section.

1.3 State of the art

Many works have been present in the area of research dealing with the coverage problem, either offline or online, aiming to accomplish static or dynamic coverage, using single or multi-robot approach, and assuming known and unknown environments. In this paragraph we will review some of the solutions found in literature as a starting point for the research developed in this thesis.

1.3.1 Coverage algorithms

Regarding static coverage in [24] is presented a distributed control strategy for deploying hovering robots with multiple downward facing cameras to collectively monitor an environment. Information per pixel is proposed as an optimization criterion for multi-camera placement problems to derive a specific cost function for multiple downward facing cameras. The cost function leads to a gradient-based distributed controller for positioning the robots, experimented with three with three AscTec Hummingbird quad-rotor robots. Although previous works have been presented previously, in which a Voronoi partition of the environment is involved, this approach to the contrary relies on the fields of view of multiple cameras to overlap with one another.

A novel approach to static coverage is presented in [23] where propose a cooperative algorithm to maximize the monitored areas in a 2D non-convex environment, by using a team of mobile robots. In particular, it deals with the maximization of an area monitored by a team of robots using vision sensors. A learning strategy is presented, able to provide a coordinated control algorithm for all the team members. In particular, the proposed approach is based on the Cognitive-based Adaptive Optimization (CAO) methodology.

Usually, regarding dynamic coverage, to find optimal paths using an offline algorithm most of the works in literature model the problem as one of the following one: *Travelling Salesman Problem*, *Chinese Postman Problem*, *Watchman route problem* or *Vehicle Routing Problem*. All of them aim to find the shortest path to cover all the vertices of a given graph by minimizing the travelled distance, but they slightly differ for the constraint they impose on the travelled path.

In [6] is firstly analysed the space decomposition problem and then, after developing a recursive strategy for completely visit a set of connected rooms, a *parallel swath* is used to perform the complete coverage of a single room, which consist in a simple back and forth exploration as shown in figure 1.1.

In [19] a series of coverage patterns are considered among which: parallel swath, contour swath with inward shift, random walk and outward spiral. In this work is claimed however that this kind of paths are not feasible for nonholonomic car-like vehicle due to the fact that the trajectories contain discontinuities in the heading which requires the vehicle to turn on the spot, but since we are using micro aerial vehicles this constraint does not apply. In the same work is also addressed the problem of covering occasional gaps and the *Travelling Salesman Problem* (TSP) is proposed as a solution imposing the constraint that the generated paths to the uncovered areas should be as short as possible to minimize overlapped coverage.

In [18] is introduced a new algorithm based on the Boustrophedon cell decomposition. The presented algorithm encodes the areas (cells) to be covered as edges of the Reeb graph. The primary contribution of their algorithm is using the solution to the CPP in order to find the optimal order, in terms of distance travelled, in which the cells are covered. Then, given the Reeb graph an *Euler tour* is calculated, that is a circuit that covers every edge in a graph exactly once. To cover the interior of a the cell a simple back-and-forth motion is used. The experiments were carried out by simulating a Pioneer robot to perform coverage of all the available free space, in different classes of environments as test cases.

In [20] is investigated the *Watchman Route Problem* (WRP), that is: find a shortest route such that each point in the polygon P is d-visible (i.e., visible and

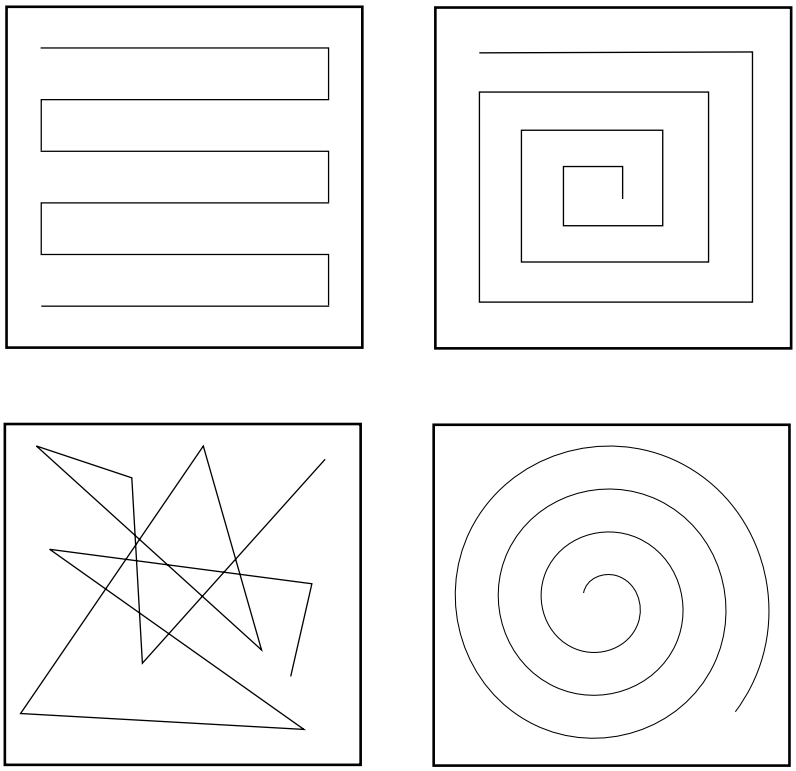

Figure 1.1: Various dynamic coverage patterns

at most d away) from some point along the route. For simple polygons when there is a visibility range d and we are interested in viewing the whole interior of P. Two versions of the WRP are presented. Find a shortest route such that either *(a)* each point in the boundary of the polygon (d-watchman problem) or *(b)* each point in the polygon (d-sweeper problem), is d-visible. route. An approximation algorithm for the TSP in simple grids that obtains solutions within 33% of the optimum. This also provides approximate solutions for the d-sweeper problem.

For the class of online algorithms, the most common approaches will be discussed more in detail in the body of the thesis, but let's firstly go briefly through all of them. The simplest online algorithm is the so called *Node Counting*. Basically this algorithm count the number of visits in the various positions of the terrain to be covered and chooses to go to the location with the smaller number of visits.

In [12] can be found a good review of the *LRTA**. The logic behind this algorithm is similar to the Node Counting one, but in this case the value associated to a map position is not just its visit count but a cost value that is used to guide the robots near the unvisited locations. For a comparison between Node Counting and LRTA* see [15].

In [16] the *Edge Counting* algorithm is presented. In this case the count used for making decisions is the edge traversal count of the departing links, from the current

position, to the next reachable ones. Random walks are bad search algorithms since they do not remember were they have already searched but it be can easily derived a real-time search algorithm that shares many properties with random walks, but has finite complexity - basically, by "removing the randomness" from random walks. In [14], is proved that edge counting always reaches a goal state with a finite number of action executions, but its complexity can be exponential in the size of the state space.

In [5] the *PatrolGraph** algorithm is introduced. It takes into account both the number of visits and as node count and the edge traversals. The algorithms has been specifically designed to solve the problem of Multi–Robot Controlled Frequency Coverage (MRCFC), in which a team of robots are requested to repeatedly visit a set of pre–defined locations of the environment according to a specified frequency distribution. It is proven to be statistically complete as well as easily implementable on real, marketable robot swarms for real–world applications.In this thesis we will exploit the properties of this algorithm to perform a Multi–Robot Uniform Frequency Coverage (MRUFC), so that all the locations of the map will be visited uniformly.

1.3.2 Space Decomposition

To perform a path planning and define a metric for deciding when a coverage is complete we have to sub-sample the space using a methodical approach. The most common approach is to apply a so called *cellular decomposition* to the space. A mathematical definition of cellular decomposition is given in definition 1.4.1.

Definition 1.3.1. ***Cellular decomposition*** In geometric topology, a cellular decomposition G of a manifold M is a decomposition of M as the disjoint union of cells (spaces homeomorphic to n-balls B_n).

Practically a cellular decomposition is a data structure that encodes the topology of a given environment, using elementary non-overlapping regions of terrain of known geometry such that adjacent cells share a common boundary and that the union of all cells coincides with the environment. The main purpose of the decomposition is to derive an abstract description of the free space, i.e., the one in which the robot can move. Moving from cell to cell and covering the space in each cell for all of them will result in covering the entire region.

We can distinguish two types of cellular decomposition:

1. Exact cell decomposition (topology-dependent)

2. Approximate cell decomposition (topology-independent)

In the exact cell decomposition the topology of the environment is accurately represented and to accomplish this all the cells does not have to be the same size, but they contain only free space. The method also requires a complete description of the objects and a complex cell construction. In approximate cell decomposition, the entire region is divided into equal sized cells. Any cell with any part of an obstacle in it is marked as invalid workspace. The cells do not necessarily have to be marked as empty or full, but can be represented by a fraction of the portion occupied. The approximation can be improved by increasing the resolution but this of course will increase the memory requirements if the entire grid is stored in memory. After the cells are created an adjacency graph can be produced where each vertex represents a cell and the edges represents the adjacency relationship between the cells. Then this adjacency graph is used by the algorithms to perform the path planning in order to complete the coverage.

In [7] the authors present some new examples of exact cellular decompositions whose cells are defined by critical points of Morse functions. Cellular decompositions have been widely used for planning a path between two points in the free space, but the motivating task for the work presented in this paper is coverage. Since the Morse functions define cells with "simple" structure, a planner can then use a cellular decomposition to achieve coverage by employing simple control strategies to cover each of the individual cells in the decomposition. A simple control strategy can be back-and-forth motions, resulting in a farming style pattern.

In [8] the Boustrophedon Cellular Decomposition is introduced (figures 1.4 and 1.5). Is an exact cellular decomposition approach, for the purposes of coverage. Essentially, the boustrophedon decomposition is a generalization of the trapezoidal decomposition that could allow for non-polygonal obstacles, but also has the side effect of having more "efficient" coverage paths than the trapezoidal decomposition. Cells are formed via a sequence of open and close operations which occur when the slice encounters an event, an instance in which a slice intersects a vertex of a polygon. There are three types of events: *in*, *out*, and *middle*. Loosely speaking, at an *in* event the current cell is closed (thereby completing its construction) and two new cells are opened (thereby initiating their construction). The boustrophedon cellular decomposition is an enhancement of the trapezoidal decomposition and is designed to minimize the number of excess lengthwise motions, as described in the

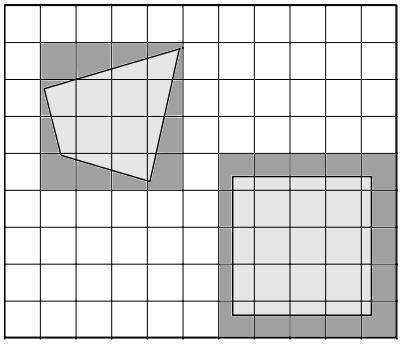

Figure 1.2: Approx. cell decomposition

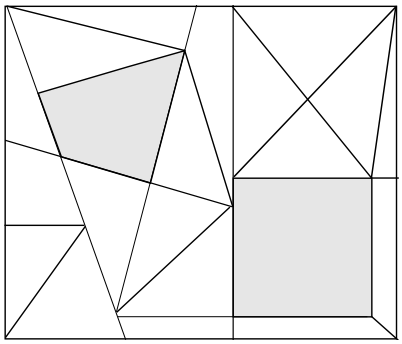

Figure 1.3: Exact cell decomposition

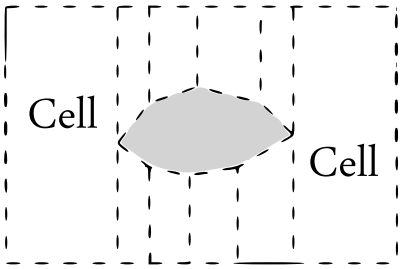

Figure 1.4: Trapezoidal decomposition

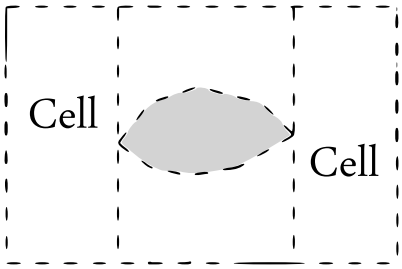

Figure 1.5: Boustrophedon decompostion

previous paragraph. In essence, all cells between *in* and *out* events are merged into one cell. The advantage of having a fewer number of cells is that to complete the coverage the number of back-and-forth boustrophedon motions can be minimized.

Regarding the approximate cell decomposition instead the logic is much simpler, since the only decision we have to make is how much we want the grid to be detailed. Then we mark as free only the cells that not contain any obstacle and all the rest as non-accessible space. In [25] a recursive strategy is presented where the cells are continuously subdivided until one of the following scenarios occurs: each cell lies either completely in free space or completely in the C-obstacle region, an arbitrary limit resolution is reached. Once a cell fulfils one of these criteria, it stops decomposing. This method is also called a *quadtree decomposition* because a cell is divided into four smaller cells of the same shape each time it gets decomposed.

Among the resulting grids we can distinguish between *simple grids* (i.e., without internal holes) and *non–simple grids* which do not possess *local cut nodes* (i.e., nodes whose removal locally disconnects the graph induced by the grid).

Both exact cell decomposition methods and approximate cell decomposition methods have advantages and disadvantages. The former are guaranteed to be complete, meaning that if a free path exists, exact cell decomposition will find

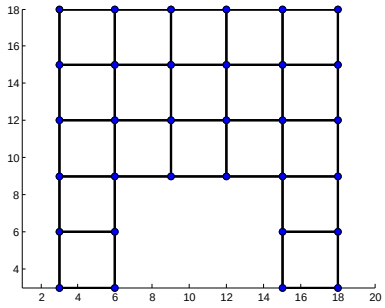

Figure 1.6: Simple grid

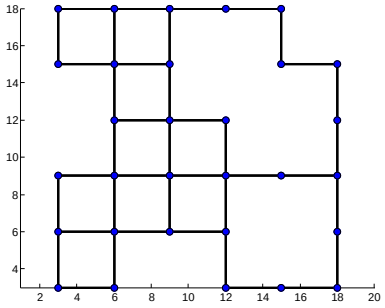

Figure 1.7: Non-simple grid w/o l.c.n.

it; however, the trade-off for this accuracy is a more difficult mathematical process. Approximate cell decomposition is less involved, but can yield similar, if not exactly the same, results as exact cell decomposition.

1.4 Terminology and Notation

Due to the simultaneous use of the ROS network and graph theory, I will always try to use two distinct names when referring to nodes and vertices. In particular in the ROS environment is formed of nodes and a graph has vertices.

That said, every terrain can be modelled as a undirected graph, by sampling the area using a grid. The grid is described by a navigation graph $G_N = (V, E)$, the *navigation graph*, where V is a set of vertices and E is a set of edges. Each edge that connects vertex i to vertex j is represented as $e_{ij} = e(i, j)$. Let be $N(v)$ the set of vertices adjacent (connected through an edge) to v. The maximum degree (number of edges incident to a vertex) of the graph G will be denoted by $\Delta(G)$ and the minimum degree by $\delta(G)$.

Definition 1.4.1. ***Eulerian State Spaces*** A state space is Eulerian if there are as many actions that leave a state as there are actions that enter the (same) state.

Since an undirected edge is equivalent to one incoming and one outgoing edge, all undirected state spaces are Eulerian.

The navigation graph is better represented through a strongly connected, oriented graph $\hat{G}_N$, derived from G_N by doubling all its edges and assigning them opposite directions. $E_i = \{e_{ij}\} \neq 0$ is the finite, nonempty set of directed edges that leave vertex $v_i \in V$. $|E_i|$ is the dimension of the set, i.e., the number of edges departing from v_i.

$R = \{r_i\}$ is a set of M robots. Robots are allowed to move in the workspace from v_i to v_j in $\hat{G}_N$ only if $e_{ij} \in E_i$, i.e., if the two vertices are adjacent. $\boldsymbol{\lambda} = [\lambda_1, \cdots, \lambda_N]^T \in \Re^N$ is a vector which describes the average visiting rate to each vertex $v_i \in S$, expressed as *number of robots* per *time unit*. $\boldsymbol{\lambda}^* = [\lambda_1^*, \cdots, \lambda_N^*]^T \in \Re^N$, $0 \leqslant \lambda_i^* \leqslant 1$ and $\sum_1^N \lambda_i^* = 1$ is a vector which describes the prescribed frequency distribution of visits.

Similarly to [15], the expression "one-of X" returns randomly one of the elements of X. The notation $succ(v, e)$ returns the vertex linked to v through the edge e. The expression $c(v)$ represents the count value associated with to the vertex v, initially set to zero for all $v \in V$.

Let be U a structure containing all the vertices of the graph still to be visited, i.e., the *unvisited set*. At last et $\ell(\cdot)$ be the function that returns the length of a path and, in particular, let $\ell(P_i) \equiv \ell_i$ be the length of the $i-$th path.

Chapter 2

Work Overview

Two main branches can be identified in the algorithms developed for the terrain coverage: online and offline search methods. This two branches can fit different needs, that are related to the type of hardware to which are applied to, i.e., micro-robots swarm or advanced-robots swarm. In particular real time search are a better choice for micro-robots that have restricted computational capability since they only have to perform a search on adjacent cells and make simple comparison choices. The optimisation algorithms implemented instead require a discrete computational capability since they analyse all the graph and compute many the possible combinations before taking the choice.
In this thesis work the following methods have been investigated:

- Online Algorithms
    1. Node Counting
    2. Learning Real-Time A*
    3. Edge Counting
    4. PatrolGRAPH*
- Offline Algorithms
    1. VRP Greedy Nearest Neighbour
    2. VRP Greedy A*
    3. VRP Greedy with Floyd-Warshall

From now on, for convenience, I will refer to the Learning Real-Time A* as LRTA*, and in the VRP Greedy I will frequently omit the *greedy* attribute.

The way the algorithms operate allow to classify them also in two other classes: *distributed* and *centralised* algorithms. In fact, while in the real-time algorithms the quadcopters are autonomous and compute their paths while moving, the optimised algorithm require a central brain that calculates the optimal paths and send them to the quadcopters. Nevertheless a distributed version of the optimised algorithm is possible to implement by distributing the evaluation of the algorithm among the robots. In the next two chapters both types of algorithms will be discussed in depth, providing a detailed explanation, pseudo-code of the algorithm, images of the generated ROS graph network and sample screen-shots of the simulator at the end of the coverage (trails are left by the quadcopters while moving).

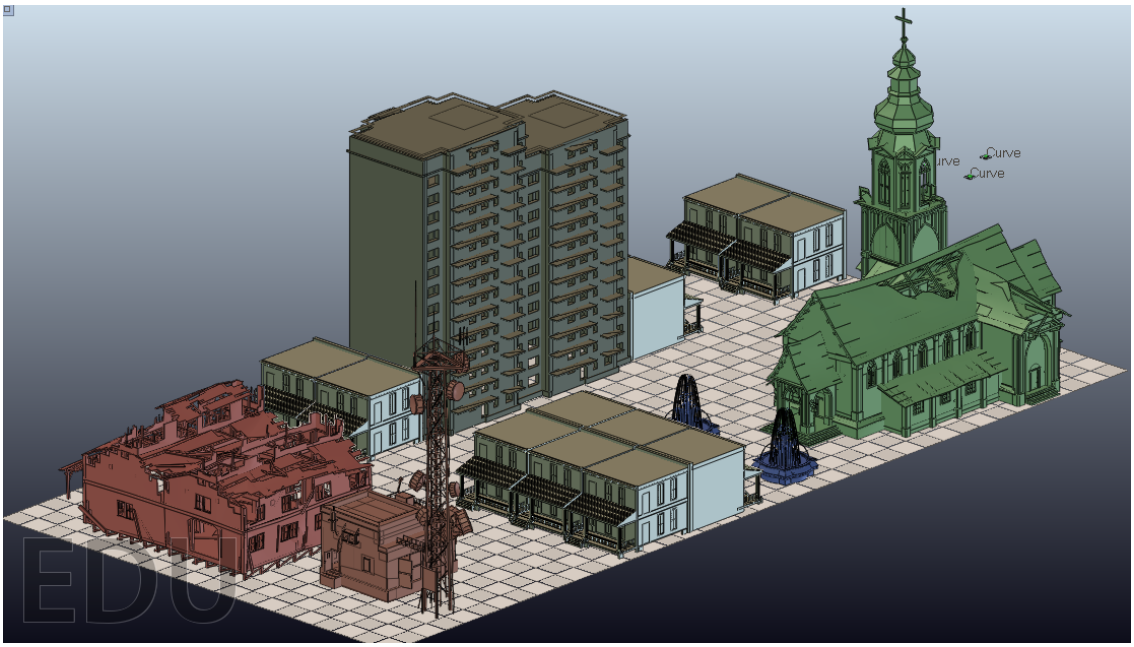

Figure 2.1: The 3D simulation environment

To test the algorithms the 3D simulation environment was created from scratch using the VREP robotic simulator. The map has been created trying to include a variety of topology characteristics, such as: dead ends, loops and open-spaces. As stated in the first chapter the environment map is sub-sampled using a regular grid, and a binary value is assigned to every cell of the grid: **0** represent an accessible cell, while **1** represents an occupied cell, as shown in figure 2.2. There exist a vertex $v \in V$ for every cell of the grid, and every vertex v is connected to its accessible adjacent w_i vertices with an undirected edge $e = (v, w_i) \in E$. The result, is a strongly connected graph, where every node v can be reached from every vertex w directly, or by traversing a finite number of vertices. The edge cost is simply calculated using the euclidean distance. The edge connecting two neighbours has

always the same cost. Nevertheless a possible extension can be applied in the search methods using the A* path finding algorithm [11]. The matrix used by the A* in fact is not binary. To an accessible cell instead, a value from 0 to 5 can be assigned, which represents the cost of travelling across that particular cell. Zero means the least possible difficulty in travelling whilst five represents the most difficult.

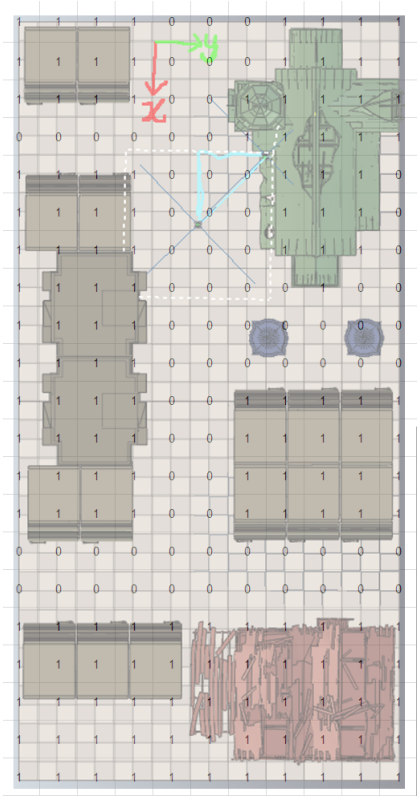

Figure 2.2: The binary access matrix built on top of the 3D environment

The input of the coverage solver is then a simple text file containing the occupancy grid matrix that the program will convert to an adjacency graph so to apply all the discussed algorithms. In this way it's really easy to describe a terrain so that having for example just an image of a geographic area we can easily convert it to an occupancy grid using some dedicated software manually or automatically. We can observe anyway that not-accessible cells of the map will never be taken into account by the algorithm since no edge points to them, but they will still occupy memory space. That's why an alternative input method has been provided for the controllers, consisting in a map already represented as an adjacency graph but containing only the accessible vertices, with an auxiliary file specifying the position in the space for each vertex. Using this representation of course the process of converting a raw image to a consistent graph becomes a little bit more complex and for small maps the reading time and memory allocation space may not vary significantly, but with

bigger and more detailed maps the advantages may become considerable. The grids in Fig. 1.6 and Fig. 1.7 for example are represented using adjacency graphs.

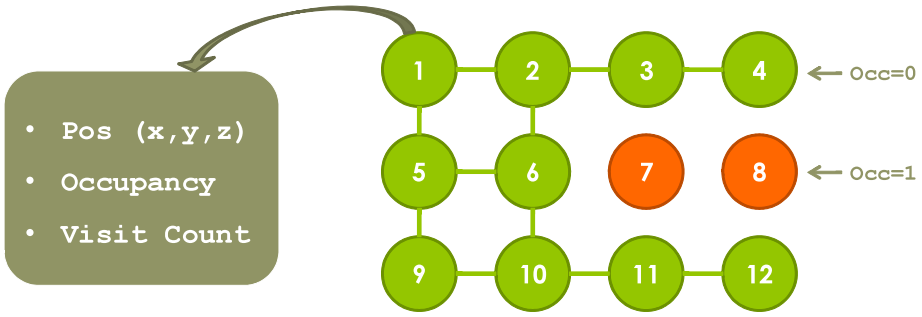

Figure 2.3: Graph structure

2.1 The way-point planning algorithm

The basic mechanism used to move the quadcopters relies on the fact that in the simulator, to each quadcopter is associated a so called manipulating sphere, shortly the MP, which the quadcopter follows whenever the sphere centre does not coincide with its centre. It comes natural then to understand that the *quad* controllers move this spheres to manipulate the robots in the environment. While performing the coverage the quadcopters move only from one vertex to a neighbour one and, under this condition no obstacle can be found if the map is accurate enough. Anyway a short remark has to be done about this: since the control algorithm embedded in the quadcopter is (as stated by the author) "quickly written and is dirty and not optimal", an additional path planning was developed beneath the search algorithms. In particular if the MP is placed too far away, the thrust given to the propellers of the quadcopter, being directly proportional to the distance MP→quadcopter, becomes too high. This causes the quadcopter to become highly unstable and eventually lose control and crash. To overcome this problem a 'critical distance' has been defined, and if the MP→quadcopter distance is greater then the critical one, intermediate way-points are dynamically interpolated to safely guide the quadcopter to the target. So combining this necessity with the one of following a given path vertex by vertex I developed the algorithm shown in figure 2.4.

In the flow-chart the way-points relative to a vertex are called targets, while the way-points between two vertices are called sub-targets. The distance between a quadcopter and the target is called *dist* while the distance between a quadcopter and a sub-target is called *subDist*. The critical distance mentioned above is shortened as *CRIT_DIST*. This argumentation does not apply to the control of the real

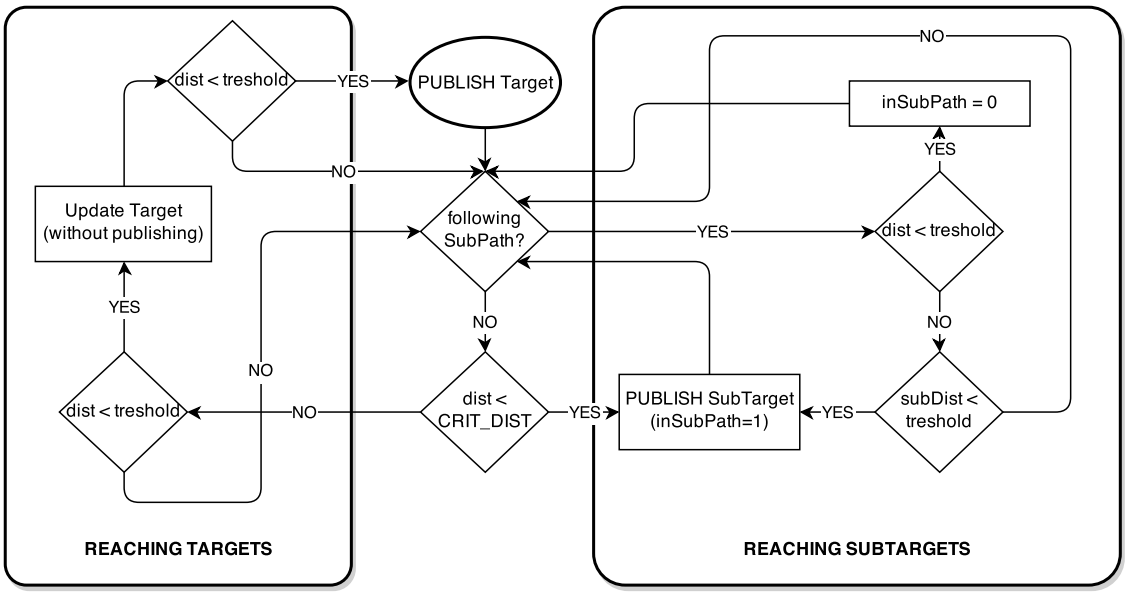

Figure 2.4: The flow-chart of the inter-vertex trajectory planning

quadcopter since the platform used for the experiments on the field. an Asctec^{TM} Pelican has a very good PID controller (better described further on) for which the targets are just sent without sub-path-planning.

All this control logic is contained within a bigger one where we check whether we acquired the quadcopter position (needed for all the distance comparisons) and if we have completed the coverage.

Chapter 3

Online Algorithms

When dealing with swarms of robots the biggest branch of studies and experiments on this topic usually refers to little robot units with limited sensing, and limited computational capabilities. In this framework simple real time search methods play a significant role since, even if the behaviour of the algorithm itself is not sophisticated, groups of robots take advantage of their collective behaviour, parallelism and fault tolerance.
A key concept in swarms behaviour is *stigmergy*, concept at the base of the path planning used in the developed real-time search algorithms. The stigmergy mechanism allow multiple agents to coordinate indirectly by leaving traces of their movements in the environment. This kind of mechanism is the same used by ants: they communicate using pheromones trails that are laid and can be followed by other ants. The more ants follow a trace, the more they refresh the pheromone left, and so other ants will follow that path more likely.

We apply the same concept but somehow in a reversed fashion. The terrain is modelled as a grid divided in cells and, since our objective is to perform a complete coverage of the grid, we make our choice by looking at the less visited cells and moving onto them. Both Node Counting and the LRTA* share the same operative work-flow which is synthesised in the following pseudo-code:

Algorithm 1 Navigation Algorithm for the real-time search

1: $v_c = v_{start}$
2: $U \leftarrow$ *"set of unvisited vertices"*
3: **while** $U \neq \emptyset$ **do**
4: $\quad e = choose(v_c, Alg)$
5: $\quad$ move along edge e
6: $\quad v_c := succ(v_c, e)$
7: **end while**

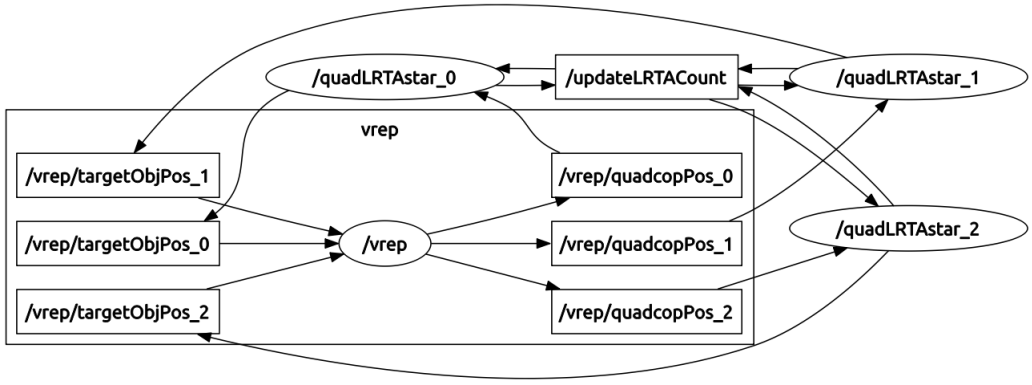

Figure 3.1: The nodes and topics ROS network of the online algorithms. The ellipses are nodes while the rectangles are topics.

The three search methods differ only in the *choose* rule which will be discussed in the next sections. The pheromone mechanism is implemented by embedding a count-grid in each agent which is updated by means of the ROS network: each node is at a same time publishing and subscribed to the same common topic *updateLRTACount*, and each time one of the agents publishes on it, all of them updates the counts in their grid.

Let's go through the algorithm: at the beginning a start vertex is assigned (line 1) to each quadcopter, and the set U of unvisited vertices is initialised (line 2). In particular the unvisited set contains all the vertices in the graph with a key value of 0, representing the accessible cells. From this vertex all the neighbour vertices w_i are tested looking at its $c(w_i)$ associated value. We can observe that this methods do not require any memory, since the update rule is just related to the vertex the robot is visiting at a given time and not on the history of its path.

All the control *quadLRTAstar* nodes are launched together using the `roslaunch` package. Each time a robot is about to move it check whether all the nodes have been visited, and if the unvisited set is empty the robots stop searching. As you can see in figure 3.1 all the ROS nodes are independent, but they communicate through the common topic. This node network is relative to the LRTA* algorithm, but the same scheme holds also for all the other online algorithms.

If a robots find itself in a situation where two adjacent vertices happen to have the same value of interest, whether this is a vertex count (Node Counting), an heuristic estimate (LRTA*), and edge count (Edge Counting) or an edge transition probability (PatrolGRAPH*), the robots chooses randomly one of the equally best values. For the multi-robot coverage due to possible communication delays the

random factor plays a significant role in spreading out the robots across the map, decreasing the total coverage time.

3.1 Node Counting

Following the logic described in the previous algorithm (1), the Node Counting search updates the vertex count using the following simple rule:

Algorithm 2 *Choose* operator for Node Counting

1: $w =$ one-of $argmin_{w_i \in N(v_c)} c(w_i)$
2: $c(v_c) = 1 + c(v_c)$
3: **return** $e(v, w)$

At the beginning of time all the vertices start with a count $c(v) = 0$, which means that they are all unvisited vertices. While the robots are exploring the area the Node Counting search make the robots move to the nearest neighbour vertex with the smallest count, so that the swarm agents are automatically steered to the closest unvisited zones. In this way after some time all the vertices will be visited.

3.2 Learning Real-Time A*

The update rule of the node Learning Real-Time A* is a little bit less trivial, since is not only based on the current vertex, but also on the successor. The LRTA* algorithm repeats the following steps until the problem solver reaches the goal state. It builds and updates a table containing heuristic estimates of the cost from each state in the problem space. Initially, the entries in the table come from a heuristic evaluation function, or are set to zero if no function is available, and are assumed to be lower bounds of actual costs. Through repeated exploration of the space, however, more accurate values are learned until they eventually converge to the actual costs to the goal [12]. Its complexity is found to be a small polynomial in n (number of states) [16]. The vertex count is updated as follows:

Algorithm 3 *Choose* operator for LRTA*

1: $w =$ one-of $argmin_{w_i \in N(v_c)} c(w_i)$
2: $c_{LRTA}(v) = 1 + c_{LRTA}(w)$
3: **return** $e(v, w)$

Let's remark first of all that the c_{LRTA} count for this algorithm is not the actual number of visits but an heuristic estimate used for the choosing phase. At the

beginning of time all the vertices start with a count $c = 0$, which means that they are all unvisited vertices. To understand the mechanism of the LRTA* search, let's imagine to be able to look at a snapshot of the vertices count in the middle of a coverage execution: the $c_{_{LRTA}}(v)$ count value of a vertex v represents the distance from this vertex v to the closest vertex w with $c_{_{LRTA}}(w)$, e.g. an unvisited vertex in a single-coverage case. Since the logic that guides the agent is always to look at the neighbouring vertices with the smallest count, the robot will not only be steered towards the nearest unvisited vertices, but in doing that they will also follow an approximately shortest path. In [15] is demonstrated how the LRTA* exhibit better performances than Node Counting for that while in the latter the cover time can be up to exponential in the number of vertices in the graph, the former guaranteed the coverage to be completed in a time polynomial to the number of vertices in the graph.

Despite this the extension of this property from the single-robot case to a multi-robot one is not straightforward and the experimental results for a single-visit coverage show how in some cases Node Counting can be more efficient. This behaviour is related to the update-count rule: while in NC a robot can communicate its vertex choice and consequently update the vertex count in the very same moment he chooses, for the LRTA* a robot has first to reach the vertex and only then it can update its count (since it's related to its next vertex choice). Due do this in LRTA* if many robots are on the same vertex nothing stops them from choosing all the same next vertex, yielding to an inefficient coverage. For single-robot coverage of course this problem does not occur.

3.3 Edge Counting

The Edge Counting algorithm the count used for making decisions is the edge traversal count of the departing links, from the current position, to the next reachable ones. Supposing we are in vertex v_c where $e_{cj} \in E_c$ form the set of its outgoing edges, the choose operator for this algorithm is:

Algorithm 4 *Choose* operator for Edge Counting

1: $l =$ one-of $argmin_{j:e_{cj} \in E_c} k_{cj}$
2: $k_{cl} = k_{cl} + 1$
3: **return** $e(c, l)$

k_{ij} is the edge count, an integer variable initialized to 0 which counts the number of times that robots have chosen to proceed to v_j after leaving v_i. In general,

no real-time search algorithm can beat the complexity of LRTA*, which is a small polynomial in n. In contrast, the deterministic Edge Counting that we derived from random walks has a complexity that is at least exponential in n. The picture changes in Eulerian state spaces. The complexity of edge counting decreases dramatically and equals the complexity of LRTA*, which remains unchanged (it even beats LRTA* in certain specific domains) [16].

3.4 PatrolGRAPH*

The logic behind the PatrolGRAPH* follows a different approach. First of all, although it is presented as online algorithm, it also presents an offline phase which is needed in order to optimise the successive online phase. So we can classify it in the middle of the two approaches. It takes into account not only information about the vertex count, but also on the edge traversal count. In particular the decision that is made in vertex v_i to choose the next vertex v_j is based on the probability associated to the edge that connects them $e(i, j)$. The algorithm is designed to solve the problem of Multi–Robot Controlled Frequency Coverage, in which a team of robots are requested to repeatedly visit a set of pre–defined locations of the environment according to a specified *frequency distribution*. In particular we want the frequency distribution to be *uniform* in order to visit all the vertices of the graph in the shortest time, and this kind of formulation is known as the Multi–Robot Uniform Frequency Coverage (MRUFC). In particular the MRUFC problem corresponds to the case when $\boldsymbol{\lambda}^*$ has a uniform distribution, i.e., for all s_i, $\lambda_i^* = 1/N$. The frequency distribution of visits in particular is controlled by tuning the so called *transition matrix* P, which contains for each edge e_{ij} the probability p_{ij} of being traversed. P is subject to the following constraints:

$$\sum_{j=1}^{N} p_{ij} = 1, \quad (i = 1, \ldots, N), \tag{3.1}$$

$$0 \leqslant p_{ij} \leqslant 1, \quad (i, j = 1, \ldots, N). \tag{3.2}$$

In its simplest form the elements of P will be:

$$p_{ij} = \frac{1}{|E_i|}, \tag{3.3}$$

if v_i and v_j are adjacent, while if s_i and s_j are not adjacent,

$$p_{ij} = 0. \tag{3.4}$$

It is easy to understand that with this kind of formulation every vertex will receive a number of visits which is proportional to its incoming edges, and we want instead all the vertices to be visited uniformly. To accomplish this the Patrol-GRAPH* performs the so called *offline phase* to tune the P transition matrix.

3.4.1 The offline phase

Let $\mathbf{1} = [1 \dots 1]^T$ be the unitary vector. The flow balance equations in matrix form, by considering the additional requirement that the sum of average visiting rates, computed over all vertices of $\hat{G}_N$, is constant, can be written as:

$$\begin{bmatrix} P^T - I \\ \mathbf{1}^T \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ C \end{bmatrix}. \tag{3.5}$$

The system (3.5) has exactly one solution whenever P is stochastic and irreducible [5]. The last equation in (3.5) allows one to determine the unique solution whose components sum up to C.

The off–line phase then has the main purpose of finding a solution to the so–called *inverse problem*, i.e., the problem of choosing P such as to guarantee that the solution of (3.5), with $C = 1$, corresponds to the prescribed frequency distribution. The off–line phase does not involve the individual robots.

Let $\mathbf{b} = [b_1 \cdots b_{N^2}]^T \in \Re^{N^2}$ be defined as a vector which parametrize the elements of the transition matrix P, so that $P = P(\mathbf{b}) \in \Re^{N \times N}$. This is done by ideally subdividing $\mathbf{b}$ into N vectors with length N, and by stacking them onto each other to build up the rows of P. More formally,

$$p_{ij} = b_{N \times (i-1)+j}, \quad (i, j = 1, \dots, N), \tag{3.6}$$

and $\boldsymbol{\lambda}^*$ represent the ideal desired solution of the MRCFC problem. Then, $\boldsymbol{\lambda}^* - \boldsymbol{\lambda}(\mathbf{b})$ is a representation of the mismatch between the actual frequency distribution and the desired one as a function of the structure of the navigation graph (expressed by the entries of matrix P).

The off–line phase of the *PatrolGRAPH** algorithm is the search for a solution to the following minimization problem with respect to the unknown variable $\mathbf{b}$.

$$\mathbf{b}^* = \arg\min_{\mathbf{b}} (\boldsymbol{\lambda}^* - \boldsymbol{\lambda}(\mathbf{b}))^T (\boldsymbol{\lambda}^* - \boldsymbol{\lambda}(\mathbf{b})), \tag{3.7}$$

subject to

$$\sum_{j=1}^{N} b_{N\times(i-1)+j} = 1, \quad (i = 1, \ldots, N), \tag{3.8}$$

$$b_{N\times(i-1)+j} \geqslant |\epsilon|, \quad (\forall i, j \quad s.t. \quad a_{ij} \in A_i), \tag{3.9}$$

$$b_{N\times(i-1)+j} = 0, \quad (\forall i, j \quad s.t. \quad a_{ij} \notin A_i). \tag{3.10}$$

3.4.2 The online phase

After performing the previous steps, the online phase follows the same pattern of the previous ones as in Algorithm 1, but now the choose operator is more elaborated:

Algorithm 5 Choose operator for PatrolGRAPH*

1: $c(v) = c(v) + 1$
2: **for all** j such that $e_{cj} \in E_c$ **do**
3: $\quad \Delta p_{cj} = (k_{cj} - \eta_j)/v_c - p_{cj}$
4: **end for**
5: $l = argmin_j(\Delta p_{cj})$
6: $k_{cl} = k_{cl} + 1$
7: **return** e_{cl}

Where: $c(v)$ same as before is the vertex count. k_{ij} is the edge count. η_j is a zero mean random variable with standard deviation σ, that is, $\eta_j = N(0, \sigma)$. In particular η is used to purposely introduce a factor of unpredictability but since we don't need this feature we will simply set it to zero. Line 1 updates the number of visits $c(v)$ received by v_c; Line 3 computes, for every adjacent vertex, the error Δp_{cj} between the ratio k_{cj}/v_c and the desired relative frequency p_{cj}; Line 5 picks the edge a_{cl} for which Δp_{cj} is minimum; Line 6 updates k_{cl}. In [5] is demonstrated that the routing policy adopted by *PatrolGRAPH** in Algorithm 5 to distribute robots along edges a_{ij} departing from s_i converges to the desired relative frequencies p_{ij}.

To demonstrate the effectiveness of the offline phase of the PatrolGRAPH* I've tested the algorithm on a 5x5 grid obstacle-free map using 3 quadcopters, running 4 simulations with the following parameters:

- No offline phase, 10 minutes patrolling (Fig. 3.2)
- With offline phase, 10 minutes patrolling (Fig. 3.3)
- No offline phase, 20 minutes patrolling (Fig. 3.4)
- With offline phase, 20 minutes patrolling (Fig. 3.5)

The figures mentioned in the list are a graphical representation of the visit count for the cells in the map. In particular the visit count is encoded as a colour and the colour-bar is the reference. First of all it is important to know that the starting point for all the quadcopters is the cell in position (2,0), and this of course affects the propagation of visit in the neighbouring cells. In Figure 3.2 and 3.4 we can observe how without running the offline phase the corner vertices, which have the lowest number of in-going edges, receive a significantly smaller amount of visits (colour near blue). In Figure 3.3 and 3.5, instead, we notice how the visit distribution is much more uniform, and as the simulation time increases the coverage becomes more uniform.

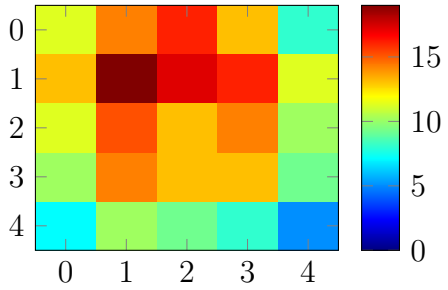

Figure 3.2: Unoptimised PG* 10 min

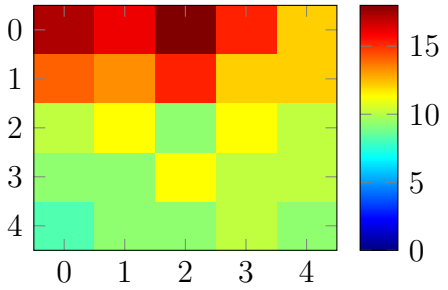

Figure 3.3: Optimised PG* 10 min

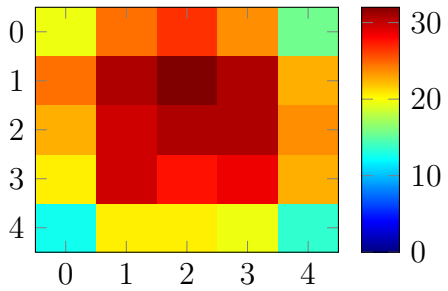

Figure 3.4: Unoptimised PG* 20 min

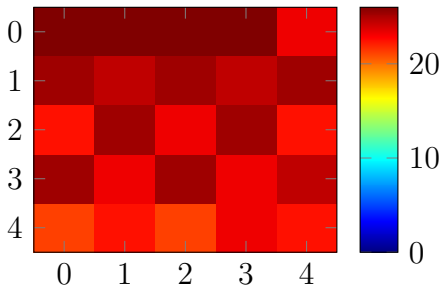

Figure 3.5: Optimised PG* 20 min

Chapter 4

Offline Algorithms

To solve the coverage problem in an optimised fashion the problem was approached as Vehicle Routing Problem, shortly VRP, using a greedy strategy which will be discussed in depth in the next paragraphs. In this case the paths are preprocessed and the robots are guided by a central ROS node which compute the optimal path. The central node is called *kernelNode* and as you can see in figure 4.1 it communicates with all the quadcopter controllers through two topics: *quadCtrlSignal* and *pathCompleted*. Same as the before all the nodes are launched together using the `roslaunch` package, but the *quadcopterRosCtrl* nodes are not able to publish their position commands until the *kernelNode* is done computing the paths. Only then the central node triggers the "start" command, and the control nodes start moving the quadcopters. When a quadcopter finishes to follow its path, it publish a message over the *pathCompleted* topic. When all the quadcopters are done, the central node stops triggering the control signal and the ROS network is shut down.

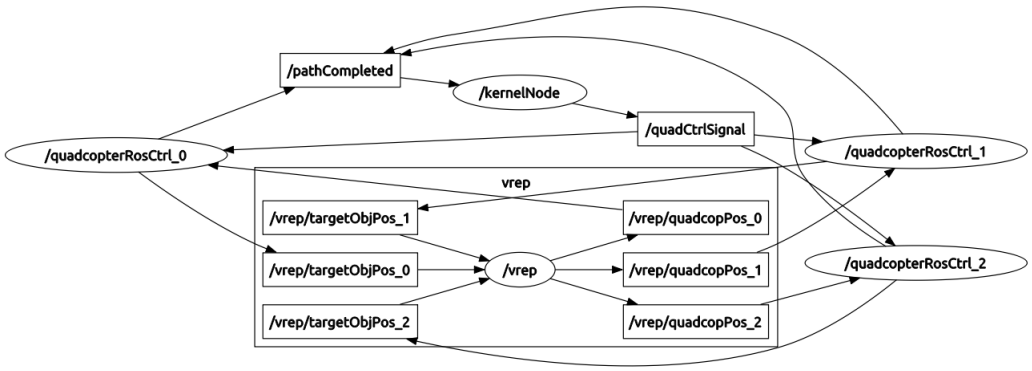

Figure 4.1: The nodes and topics ROS network of the offline algorithms. The ellipses are nodes while the rectangles are topics.

The power of this algorithm is that the paths created are cycles so that at the end of the execution all the robots are back in the beginning position, which can be useful from a logistic point of view in a urban search and rescue application, and it's convenient from an energetic point of view. For a survey on *green* VRP see [4]. To get to the final version of the VRP Greedy I went through some intermediate solutions that I will describe in the following section to show the evolution of algorithm. The first version was just using nearest neighbour nodes when constructing the path, but in a multi-robot approach this turned out to be highly inefficient. So the A* algorithm came in to take into account all the vertices still to be inserted and using the shortest path algorithm to find a way to non-neighbouring vertices. But when increasing the size of the maps computing the A* algorithm each step of the Greedy alg. became time consuming. So finally, since the map is known a priori the Floyd-Warsall algorithm became the most natural solution to find the shortest paths between all pair of vertices at once and this drastically increased the execution time.

4.1 min max **Vehicle Routing Problem**

Let's consider the problem of exploring an area with a swarm of quadcopters.

Let d_{vw} be the distance between the vertex v and the vertex w. We can assume that d_{vw} is the euclidean distance between the centre of the cell corresponding to the vertices v and w. Let $F = \{1, \dots, i, \dots, N\}$ be the set of quadcopter and let s_i be the starting vertex of the quadcopter i. A path P_i is assigned to each quadcopter, consisting in an ordered sequence of vertices of V to be visited. In particular the path of each quadcopter i starts in s_i and ends in e_i. Let $\ell(\cdot)$ be the function that returns the length of a path and, in particular, let $\ell(P_i) \equiv \ell_i$ be the length of the $i-$th path.

The problem consist in planning the trajectories such that:

1. the distance travelled by the quadcopter with the longest path is minimized

2. every vertex is visited at least once

In symbols:

$$\min \quad \mathcal{L} \tag{4.1}$$

subject to:

$$\mathcal{L} \geqslant \ell(P_i) \quad \forall i \in F \tag{4.2}$$

$$v \in \cup_{i \in F} P_i \quad \forall v \in V \tag{4.3}$$

Where (4.1) is the *objective function*, the constraints (4.2) impose $\mathcal{L}$ as maximum length and the constraints (4.3) impose that every node is visited.

4.2 The Greedy Algorithm

The greedy algorithm is a constructive procedure that iteratively produces a solution. At each iteration the algorithm takes an incomplete solution (potentially empty) and adds a vertex to produce a new solution. At every step the algorithm finds the vertex v to be inserted, together with the path i and the position p in which the vertex has to be inserted.

In particular there are two nested levels of optimisation. The first one is path-local and, for a given robot i, checks among the different insertion positions p of a same vertex v which one generates the shortest tentative path length. The second level of optimisation instead is global and checks among all the shortest P_i paths which one generates the smallest increment with respect to the current longest path. The current length of the longest path will be identified with the letter $\mathcal{L}$. The choice is made by testing the δ_{vip} increment in the objective function and then choosing the configuration $< v, i, p >$ that produces the smallest increment. The complexity of the algorithm is then polynomial $O(|V|^2 N)$.

In the basic version is convenient to use a single variable δ and a single triplet $< v, i, p >$ to be updated at every iteration. We will call $P_i^{<v,i,p>}$ the path P_i modified inserting the vertex v in the position p. It is convenient, for an implementation point of view, to save the tentative path $P_i^{<v,i,p>}$ in a variable different from P_i (it is enough just a single temporary path instead of one for each vehicle).

Here is the algorithm in detail: the rows 1–8 are used to initialise all the variables. In particular the structure of U is initialised (line 1) with the set of all the accessible nodes (not shown in the code: the start nodes s_i are excluded). The paths are initialised with a sequence that begins and ends with the starting nodes (2–4). At the beginning of the main loop (line 5), the variable δ_{vip}, containing the increments of the objective function is initialised at plus infinite, and the same thing goes for the best increment δ^{best} (line 6). We then initialise $\mathcal{L}$, with the length of the longest path and reset the best choice (lines 7–8). In the next three nested loops (lines 9–24) is tested the insertion of every vertex $v \in U$ in every possible position p of every possible path i. For a given robot i, the length after the insertion, called ℓ_i^{tent}, is then compared to the smallest current length among all the insertions, $\ell_i^{\min}$ (line 14). Note: the position to consider are the ones included between the position after the start and before the end. If the tentative length is smaller than the current best

Algorithm 6 Greedy algorithm for the min max VRP

1: $U \leftarrow V$
2: **for all** $i \in F$ **do**
3: $\quad P_i \leftarrow (s_i, e_i)$
4: **end for**
5: **while** $U \neq \emptyset$ **do**
6: $\quad (\delta_{vip}, \delta^{\text{best}}) \leftarrow +\infty$
7: $\quad \mathcal{L} = \max_{\forall i \in F} \ell(P_i)$
8: $\quad$ choice $\leftarrow \emptyset$
9: $\quad$ **for all** $v \in U$ **do**
10: $\quad\quad$ **for all** $i \in F$ **do**
11: $\quad\quad\quad (\ell_i^{\min}, \ell_i^{\text{tent}}) \leftarrow +\infty$
12: $\quad\quad\quad$ **for all** position $p \in P_i$ **do**
13: $\quad\quad\quad\quad \ell_i^{\text{tent}} = \ell(P_i^{<v,i,p>})$
14: $\quad\quad\quad\quad$ **if** $\ell_i^{\text{tent}} < \ell_i^{\min}$ **then**
15: $\quad\quad\quad\quad\quad \ell_i^{\min} = \ell_i^{\text{tent}}$
16: $\quad\quad\quad\quad\quad \delta_{vip} = \ell_i^{\min} - \mathcal{L}$
17: $\quad\quad\quad\quad\quad$ **if** $\delta_{vip} < \delta^{\text{best}}$ **then**
18: $\quad\quad\quad\quad\quad\quad \delta^{\text{best}} \leftarrow \delta_{vip}$
19: $\quad\quad\quad\quad\quad\quad$ choice $\leftarrow < v, i, p >$
20: $\quad\quad\quad\quad\quad$ **end if**
21: $\quad\quad\quad\quad$ **end if**
22: $\quad\quad\quad$ **end for**
23: $\quad\quad$ **end for**
24: $\quad$ **end for**
25: $\quad$ add v to P_i in position p
26: $\quad U \leftarrow U \setminus choice.v$
27: **end while**
28: **return** $P = \cup_{i \in F} P_i$

one it becomes the new best (line 15) and the increment is calculated with respect to the objective function previously evaluated, using the following formula:

$$\delta_{vip} = l_i^{\min} - \mathcal{L} \tag{4.4}$$

The best $< v, i, p >$ triplet (line 17) is saved (line 19) together with the corresponding increment (line 18). After all the for loops are executed the best insertion is applied (line 25) and the chosen vertex is removed from the list of unvisited vertices (line 26). When the unvisited set U is empty (line 27), the solution is returned (line 28).

4.3 VRP Greedy Nearest Neighbour

This version of the VRP Greedy follows basically the exact same scheme explained in the previous paragraph. The reason why it was called the "Nearest Neighbour" comes from the way it looks for the vertices. The candidates vertices to be inserted in a the path of a robot i are chosen among all the vertices in the neighbouring of that robot's path P_i. Remember that by how we defined our graph in the first chapter (1.4), a vertex v has edges that point only to its neighbours w_i, so any other vertex insertion would have been inconsistent. So if we are to test vertex v_t in positon p of path P we only insert it if $e(P(p-1), v_t) = 1$ and $e(P(p-1), v_t) = 1$. If we were to operate with a single robot, it always true that there exist a neighbour unvisited vertex, but this is no more valid when we deal with a swarm of robots. To solve this problem the VRP Greedy A* was developed.

4.4 VRP Greedy A*

This extension of the previous VRP make possible for the algorithm to check vertices which are not in the neighbouring of a certain P_i path. When all the neighbour vertices of a certain path have been already visited, the greedy search starts looking for all the remaining vertices, and it finds a way through the graph using the A* path-finding algorithm. In particular if we consider the insertion of a vertex v in a position p, the destination position will be between the vertex $P(p)$ and $P(p-1)$. The algorithm checks firstly whether the vertex v is a neighbour of $P(p-1)$ and, if not, it calculates the so called "*way there*". Then it checks whether the vertex v is a neighbour of $P(p)$ and, if not, it calculates the so called "*way back*". The final A* path will be then:

$$waythere + v + wayback \tag{4.5}$$

To fit the behaviour of this algorithm the $< v, i, p >$ variable was modified by introducing a fourth parameter n, a boolean variable which tells if the current best choice is a vertex adjacent to the corresponding P_i path or not. The resulting new choice was simply called $< v, i, p, n >$, and a the moment of the insertion (line 25 of algorithm 6), if $n =$ `false` the A* path of will be inserted in position i.

The problem found with this approach was that the A* algorithm is performed every time a vertex v is tested on a position p and since the greedy algorithm tests all the possible configuration, on maps with more than 100 nodes the computational time started to be too big. Moreover if assume the map is not changing, which we

are, there is no need of computing the shortest path every time, and that's why the next algorithm was developed.

4.5 VRP Greedy with Floyd-Warshall

The Floyd-Warshall algorithm is a method to find the lengths of the paths between all pairs of vertices. Using an auxiliary *parent* structure it also possible to to get the actual path, needed for the final evaluation of the optimal coverage path. The Floyd-Warshall algorithm falls in the category of *dynamic programming* since it breaks down the problem of finding a path between two arbitrarily distant vertices into simpler sub-problems.

The algorithm maintains a distance matrix (D) such that at iteration k, d_{ij} is the shortest path from i to j using nodes 1, 2, ...k as intermediate nodes. After the algorithm terminates, assuming that no negative cost cycle is present, the shortest path from nodes i to j is d_{ij}. P is the parent matrix, and the meaning of its entries p_{ij} is: "for some shortest path from $i$to j, the vertex right before j is p_{ij}". So given a graph $G(E, V)$, the algorithm proceeds as follows:

Algorithm 7 Floyd-Warshall Algorithm

```
1: D ← G
2: P ← G
3: for all i ∈ G do
4:    for all j ∈ G do
5:       if i == j ∨ d_ij == ∞ then
6:          p_ij = −1
7:       else
8:          p_ij = i
9:       end if
10:   end for
11: end for
12: for all i ∈ G do
13:   for all j ∈ G do
14:      for all k ∈ G do
15:         dist_ikj = d_ik + d_kj
16:         if dist_ikj < d_ij then
17:            d_ij = dist_ikj
18:            p_ij = p_kj
19:         end if
20:      end for
21:   end for
22: end for
23: return D
```

We initialize all entries p_{ij} to i when we have an edge, and if not we set it to -1. The update process of the parent array basically mean the following: "if going from i to j through k ($i \to k \to j$) is an improvement, then we set the parent of j in the new shortest path to the parent of the path $k \to j$". To recover the path from i to j simply recurse on p_{ij}'s entries. In this formulation inside the triple loop of the greedy solver the vertices inserted in the path are just the one in the unvisited set U, without all the shortest paths between not adjacent nodes. The shortest paths all inserted all at once at the end of the execution (post-insertion), and not while executing the solver (contextual-insertion). This reduces a lot the number of possible insertion positions $p \in P_i$ increasing the algorithm execution speed, but also meaning that we reduce the amount of solution space we explore. Some comparisons between result with the shortest path contextual-insertion and the post-insertion showed that there was so significant improvement to justify the former approach in favour of the latter.

In the next chapter will be presented the results obtained for the different algorithms and due to the motivation explained in this chapter the only algorithm used for the offline class will be the VRP Greedy using Floyd-Warshall.

Chapter 5

Simulations and Results

The five algorithms where tested on different kind of maps and compared on three major performance indexes. Since our goal is to perform the coverage in the shortest time possible, minimize the total energy spent and also don't have redundant robots that do not produce a significant contribution to the coverage, the comparison criteria are based on:

- Longest path (among all the robots) - $\max(\ell_i)$
- Total length of paths (sum over all the robots) - $\sum(\ell_i)$
- Standard deviation of the paths lengths - $\sigma(\ell_i)$

The experimental phase is divided in two sections. In the first phase we analyse the algorithms behaviour in maps with random obstacles and compare the performances. The aim is to check among the different algorithms which one performs better in each situation. In the second phase instead the aim is to identify the optimal number of robots for a given map size so all the algorithm are tested varying the number of robots. The maps considered in this second phase are obstacle-free. For a navigation graph $G_N(E, V)$ the standard coverage is said to be completed when all the vertices have been visited at least once, in formulas:

$$\min_i c(v_i) \geqslant 1 \qquad \forall v_i \in V \tag{5.1}$$

Since some online algorithms due to their nature exhibit a more efficient behaviour on multiple coverages, i.e. patrolling, these algorithms have been tested also for $\min c \geqslant C$ where C is greater than one.

For the first phase on grid maps with random obstacles, let's differentiate the three types of maps used in the algorithm by defining an obstacle index ω. The reference figures are Fig. 5.1, Fig. 5.2 and Fig.5.3.

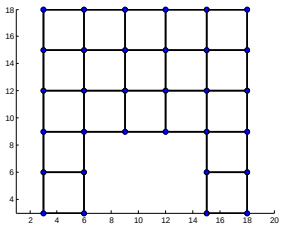

Figure 5.1: ω=1

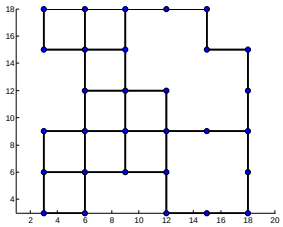

Figure 5.2: ω=2

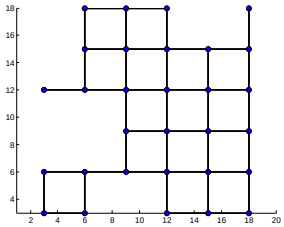

Figure 5.3: ω=3

Basically for $\omega = 0$ we have an obstacle-free map. For $\omega = 1$ we have a map without internal holes and $\delta(G) = 2$. For $\omega = 2$ we have a map with internal holes and still $\delta(G) = 2$. For $\omega = 3$ we have a map with internal holes and $\delta(G) = 1$. Tp test the algorithms the following simulation were performed:

Set 1. Increasing map size: 4x4 Map, 6x6 Map, 8x8 Map

Set 2. Increasing ω: $\omega = 1$, $\omega = 2$, $\omega = 3$

Set 3. Increasing number of robots: 2 robots, 4 robots, 6 robots

Set 4. Increasing number of min-count per vertex: $\min c = 1$, $\min c = 2$, $\min c = 5$, $\min c = 10$

For each grid five different samples where tested and the result reported in the tables is the mean of the five simulations.

In the second phase to identify the best number of robots, three grid-maps of increasing size were tested, in particular the grids are: 5x5, 10x10, 15x15. For all the grids ten different simulations were performed increasing the number of quadcopters used, from a single robot coverage to a multi-robot coverage with 10 agents. In the following figures are presented 2 examples of coverage.

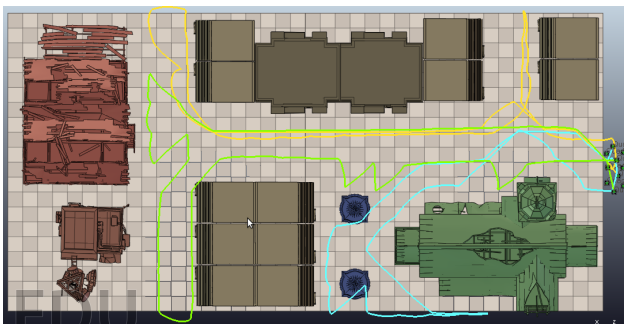

Figure 5.4: City map using VRP w/ F.W.

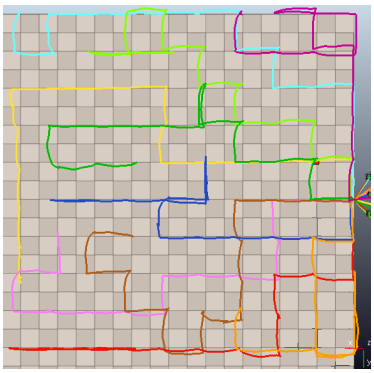

Figure 5.5: Map 10x10 using N.C.

5.1 Algorithms Comparison

5.1.1 Single robot behaviour

Before jumping to the multi-robot case let's make some considerations on the single-robot coverage. The extension of a single-robot coverage algorithm to the multi-robot case is not straightforward and at a first glance some of the results in the next sections may appear counterintuitive. For example as stated before LRTA* suffers a performance degradation in the multi-robot case. So, even if the scope of this thesis is focused on multi-robot coverage, to prove that the algorithms developed are fully functional and have a behaviour congruent with the result found in literature, simulations were performed also for a single-robot configuration using a 4x4 grid map.

Since an interesting trend can be observed when changing the number of minimum visits per vertex we tested the algorithms for 4 different values of $\min c$. The results are presented in table 5.1.

Algorithm	$\min c$	$\ell(P)$
Node Count	10	178
	5	98
	2	51
	1	17
LRTA*	10	160
	5	79
	2	35
	1	17

Algorithm	$\min c$	$\ell(P)$
Edge Count	10	250
	5	134
	2	75
	1	34
PatrolGRAPH*	10	178
	5	108
	2	57
	1	47

Table 5.1: Single-robot comparison for a 4x4 map and $\omega = 0$

As we can see the LRTA* always perform equal or better than Node Counting. It is interesting to notice how approaching the single coverage, the difference between NC and LRTA* becomes less considerable, becoming eventually equally for $\min c = 1$. Comparing Edge Count and PatrolGRAPH* we can see how for $\min c > 1$ the latter always completes the patrolling in fewer steps, while for the single coverage instead Edge Counting turns out to be more efficient. This happens since, as can be observed in appendix A, in the first steps of the algorithm PG* tends to make the robot go back to already visited vertices. The reason behind this behaviour relies in the fact that having computed a Probability Transition Matrix that ensures every vertex to be visited with a probability $p = 1/N$, PG* in the initial transient phase spends more time visiting vertices with a low number of edges to

make coincide the desired visit frequency with the actual one. Apart from this we can notice how the paths length for NC and LRTA* are always smaller than the ones resulting from EC and PG*, and this is also true for the multi-robot case as we will see in the next sections, making clear that vertex-count oriented algorithms are more suitable for coverage purposes.

5.1.2 Node Counting

In table 5.2 are presented the results for the Node Counting algorithm.

Set	Parameters				Results		
	ω	#Rob	Grid Size	$\min c$	$\max(\ell_i)$	$\sum(\ell_i)$	$\sigma(\ell_i)$
1	1	3	4x4	1	7	19	0.394
	1	3	6x6	1	20	60	0.673
	1	3	8x8	1	37	109	0.785
2	1	3	6x6	1	20	58	0.605
	2	3	6x6	1	16	45	0.625
	3	3	6x6	1	19	55	0.394
3	1	2	6x6	1	27	54	0.341
	1	4	6x6	1	14	54	0.615
	1	6	6x6	1	11	60	0.770
4	1	3	6x6	10	114	338	1.414
	1	3	6x6	5	68	202	0.577
	1	3	6x6	2	30	89	0.816
	1	3	6x6	1	15	44	0.816

Table 5.2: Results of the simulation for the Node Counting Alg.

Despite it is the simplest of the algorithms considered it is surprisingly good for multi-robot coverage. For the simulation in Set 1 of increasing map size with $\omega = 1$ the LRTA* is the only other algorithm with similar performances, although in the 8x8 grid NC still performs better. Also for Sets 2 and 3 the two algorithms get similar results, but for the 6 robots case (Set 3), the Node Counting completes the coverage in fewer steps.

5.1.3 Learning Real-Time A*

In table 5.3 are presented the results for the Learning Real-Time A* algorithm. As stated in section 3.2 the update-rule of LRTA* is not triggered till the robot reaches the target vertex and this entails that multiple robots can make the same

choice consequently not contributing in spreading the robots across the map. This disadvantage clearly emerges from the simulations where in almost all the cases the Node Counting achieves better results. This becomes even more noticeable in Set 4 for big values of $\min c$ where even if the difference between $\max(\ell_i)$ is not that big, thinking in terms of total energy spent by the swarm, $\sum(\ell_i)$, in LRTA* is significantly bigger (NC 338 vs. LRTA* 456).

Set	Parameters				Results		
	ω	#Rob	Grid Size	$\min c$	$\max(\ell_i)$	$\sum(\ell_i)$	$\sigma(\ell_i)$
1	1	3	4x4	1	7	24	0.558
	1	3	6x6	1	19	56	0.773
	1	3	8x8	1	43	125	0.558
2	1	3	6x6	1	23	68	0.762
	2	3	6x6	1	20	60	0.490
	3	3	6x6	1	16	46	0.840
3	1	2	6x6	1	26	50	0.624
	1	4	6x6	1	19	72	0.387
	1	6	6x6	1	15	85	0.725
4	1	3	6x6	10	154	456	1.633
	1	3	6x6	5	96	286	1.00
	1	3	6x6	2	31	91	0.577
	1	3	6x6	1	18	53	0.816

Table 5.3: Results of the simulation for the LRTA* Alg.

5.1.4 Edge Counting

The first thing we notice from the results in table 5.4 is that average number of visits, and consequently also total length, of the Edge Counting algorithm is generally higher for any of the 4 test cases with respect to the previous to algorithms. The results are more in line with PatrolGRAPH* algorithms ones, also for what regards the $\sigma(\ell_i)$ that in Set 4 reaches values up to $\sigma = 4.08$, while for the same set the maximum for LRTA* is $\sigma = 1.6$. In general it comes out that for a single-coverage task (Sets 1-2-3) on average the Edge Counting is faster than PG*. It is worth reminding anyway that in none of the previous coverage algorithm we have control over the frequency distribution of visits, so it is not possible for example to define an "region priority", which can instead be achieved using the PatrolGRAPH*.

Set	Parameters				Results		
	ω	#Rob	Grid Size	$\min c$	$\max(\ell_i)$	$\sum(\ell_i)$	$\sigma(\ell_i)$
1	1	3	4x4	1	18	52	0.605
	1	3	6x6	1	69	203	1.15
	1	3	8x8	1	142	422	1.32
2	1	3	6x6	1	59	175	1.04
	2	3	6x6	1	56	167	0.860
	3	3	6x6	1	46	133	0.958
3	1	2	6x6	1	75	148	0.974
	1	4	6x6	1	44	171	0.800
	1	6	6x6	1	42	244	1.06
4	1	3	6x6	10	205	605	4.08
	1	3	6x6	5	145	430	2.38
	1	3	6x6	2	75	222	0.816
	1	3	6x6	1	63	188	0.816

Table 5.4: Results of the simulation for the Edge Counting Alg.

Set	Parameters				Results		
	ω	#Rob	Grid Size	$\min c$	$\max(\ell_i)$	$\sum(\ell_i)$	$\sigma(\ell_i)$
1	1	3	4x4	1	25	74	0.662
	1	3	6x6	1	65	192	0.908
	1	3	8x8	1	150	447	1.17
2	1	3	6x6	1	61	182	0.836
	2	3	6x6	1	66	193	1.19
	3	3	6x6	1	69	204	1.39
3	1	2	6x6	1	88	174	0.832
	1	4	6x6	1	51	201	0.852
	1	6	6x6	1	41	241	0.539
4	1	3	6x6	1	180	533	3.366
	1	3	6x6	2	114	339	0.816
	1	3	6x6	5	70	207	0.816
	1	3	6x6	10	66	198	0

Table 5.5: Results of the simulation for the PatrolGRAPH* Alg.

5.1.5 PatrolGRAPH*

Observing the PatrolGRAPH* results in table 5.5 we find that the values for Set 3 are against the trend of all the previous algorithms. While previously for growing ω

the $\max(\ell_i)$ index is decreasing, in the PG* case smaller $\delta(G)$ cause longer coverage paths. This behaviour can be explained with the same reasoning done for the single-robot case in section 5.1.1 recalling that the PG* algorithm will spend more time on vertices with small number of edges to ensure that on a short term time horizon the expected and actual frequency distribution concide. It is worth noting that on the long run the PG*, ensuring a uniform frequency distribution, performs better than Edge Counting as we can see the case of $\min c = 10$ of Set 4.

5.1.6 VRP Greedy with Floyd-Warshall

For the reasons explained in the previous chapter regarding computation time and quality of the solutions, the VRP Greedy with Floyd-Warshall is the only offline algorithm that will be used in the simulations. The results are presented in table 5.6. Since the result of the Greedy algorithm is deterministic there is no point in considering $\min c > 1$, as the total path length can be easily evaluated by multiplying the result indexes of the single-coverage case by the number of coverages.

Set	Parameters			Results		
	ω	#Rob	Grid Size	$\max(\ell_i)$	$\sum(\ell_i)$	$\sigma(\ell_i)$
1	1	3	4x4	10	25	1.72
	1	3	6x6	20	53	2.10
	1	3	8x8	33	90	2.52
2	1	3	6x6	17	48	1.39
	2	3	6x6	19	51	2.32
	3	3	6x6	20	55	1.70
3	1	2	6x6	24	45	1.80
	1	4	6x6	16	59	1.27
	1	6	6x6	15	67	4.18

Table 5.6: Results of the simulation for the VRP w/ F.W. Alg.

Comparing the VRP results with the Node Counting ones, which has been found to be the simplest yet the best algorithm so far, we can observe how Node Counting is surprisingly powerful. There are few cases for which VRP is better than NC, and this occur occur mainly with big maps (8x8 grid) and low ω. We can observe the same results also for the simulation in the next section, and deduce then that the Greedy VRP performs better in obstacle-free maps, with a bigger "manoeuvring space".

5.1.7 Comparison Conclusions

From the analysis carried in this chapter it emerged how on average the best solution for the single-coverage multi-robot problem is produced by the Node Counting algorithm. First of all is interesting to notice that a (simple) offline method (no heuristic post-optimisation), does not necessarily produce a better solution than a (simple) online method. A very basic swarm rule (NC) can achieve better results than the a fairly complex routing algorithm and also, being an online method, NC is more flexible and suitable for autonomous distributed systems than VRP. This result against the trend of the results found in literature for the single-robot coverage makes clear how the extension of current coverage approaches to the multi-robot case is not trivial and requires an accurate study on communication protocols between robots and collective behaviour in general.

5.2 Finding the optimal number of robots

To find the optimal number of robots the map tested were: the city environment map and 3 increasing size obstacle-free square maps: 5x5, 10x10, 15x15. After normalising the result parameters, the method used for establishing the optimal number of robots is evaluating the output of a weighted loss function $\mathscr{L}$ where three λ weight are assigned to each of the result parameters. In formulas, for a generic number of robots R:

$$\mathscr{L}_R(\lambda, \ell_i) = \lambda_1 \cdot \max(\ell_i)_N + \lambda_2 \cdot \sum(\ell_i)_N + \lambda_3 \cdot \sigma(\ell_i)_N \tag{5.2}$$

Where subscript N stands for normalised, and the best configuration will be the one that satisfies the following formula:

$$R_{best} = \arg\min_R \mathscr{L}_R(\lambda, \ell_i) \tag{5.3}$$

By acting on the parameters we can set a lexicographical order that meets the objective we want to achieve. The λ parameters where obtained with a simple tuning based on the fact that the most important parameter is λ_1 since it weights $max(\ell_i)$ that defines the coverage completion time, and we want to minimize it. Then on a second level we want that the total energy spent by the swarm is minimized and at last we don't want a big disparity between paths lengths since it means that are some redundant robots. The parameters of the online and offline algorithms are not the same since in the online algorithms, till the coverage is completed, each robot performs the same number of steps, so λ_3 was set to a very small value with respect to the other two parameters. By taking advantage of the analysing performed in the previous section the simulations for this case will be run only for the Node Counting and VRP algorithms. In the following figures all the values in the graphs will be also represented as normalised. Due to the long execution times of the Node Counting (the deterministic solution of the Greedy is almost instant), the two algorithms will be compared for maps sizes not greater than 15x15.

5.2.1 Node Counting

A common trend in all the plots is that the maximum path length $\max(\ell_i)$ always decreases with the increasing number of robots, which is an expected intuitive result. Nevertheless there is point where increasing the number of robots is no more convenient due to the consequent increase of the total paths length $\sum(\ell_i)$. An interesting result that can be observed in the profile of the total length curve is that

there are some local minima that for larger sizes of the map occur in correspondence of a higher number of robots.

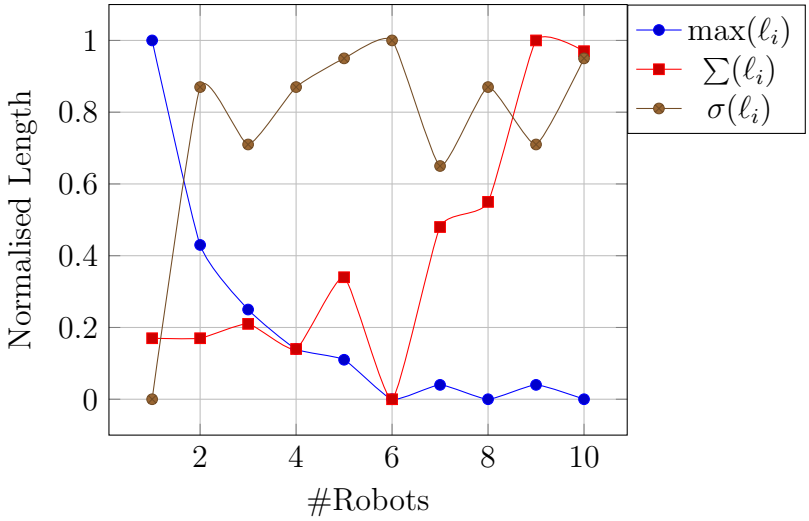

Figure 5.6: Performance indexes increasing the num. of robots, 5x5 grid using NC

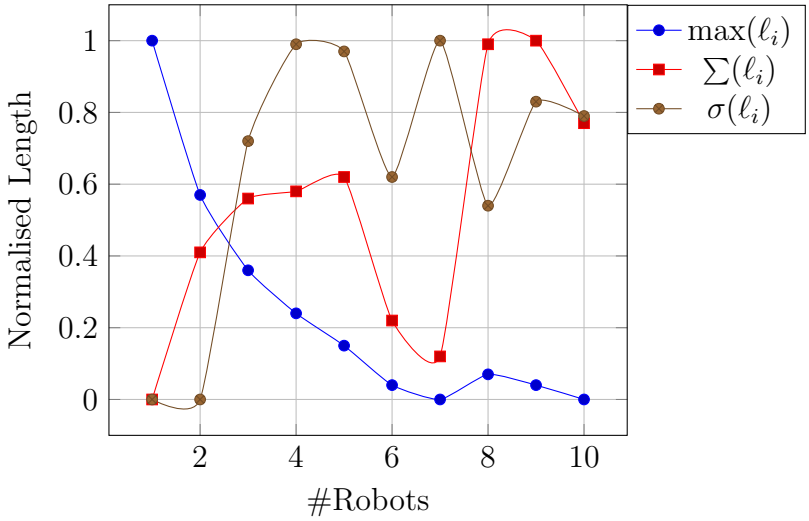

Figure 5.7: Performance indexes increasing the num. of robots, 10x10 grid using NC

So for the Node Counting algorithm we have that for a 5x5 map the best number of robots is 6, for the 10x10 map is 7 and for the 15x15 map is 10. The bigger leap between the last two results derives from the fact that a linear increase in the map side obviously produces a quadratic increase in the map area, thus on the size of the navigation graph. Even though no precise pattern is emerging from the relationship between number of robots and size of the map, this results give anyway an idea about the order of magnitude of the robots necessary for coverage for increasing map size.

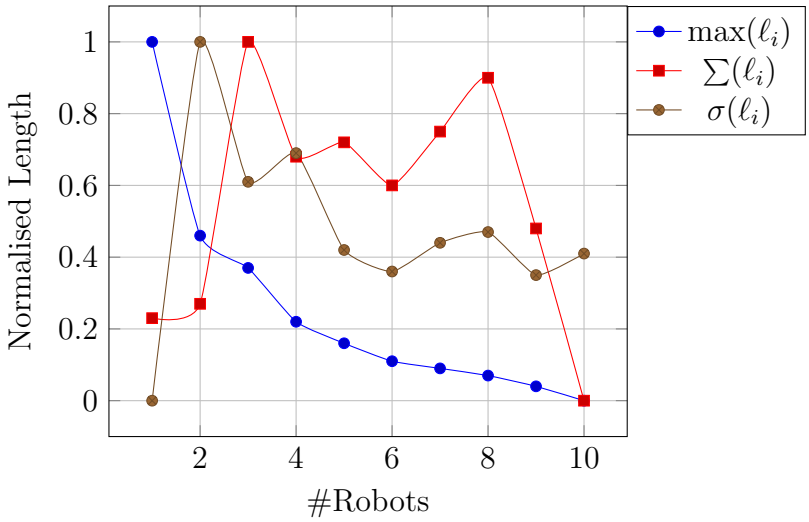

Figure 5.8: Performance indexes increasing the num. of robots, 15x15 grid using NC

5.2.2 VRP with Floyd-Warshall

In the following figures are plotted the results for increasing number of robots and increasing map size using the VRP Greedy coverage algorithm.

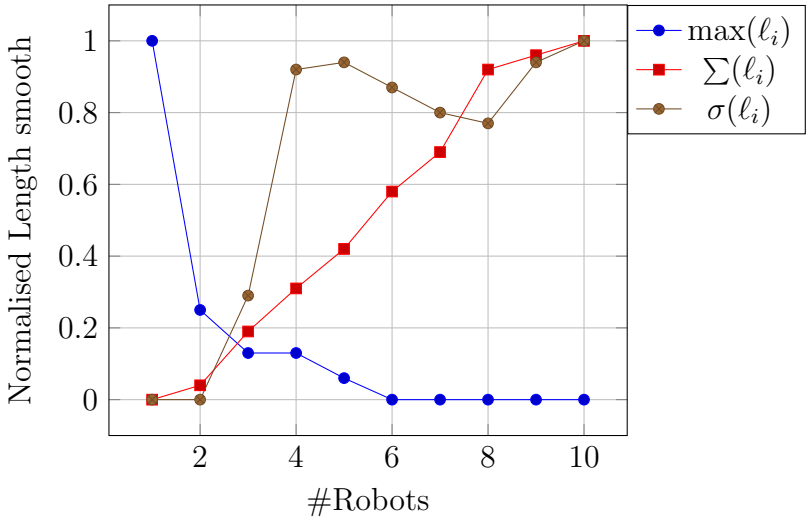

Figure 5.9: Performance indexes increasing the num. of robots, 5x5 grid using VRP

For the VRP algorithm we find that for a 5x5 map the best number of robots is 2, for the 10x10 map is 3 and for the 15x15 map is 5. We can clearly see how using this kind of algorithm the best number of robots is on average less than the one needed for Node Counting, that is a desirable property in term of resource saving.

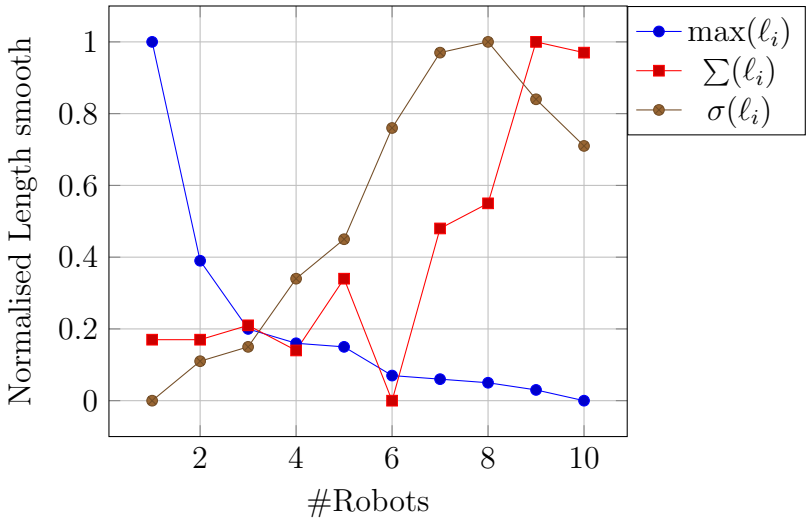

Figure 5.10: Performance indexes increasing the num. of robots, 10x10 grid using NC

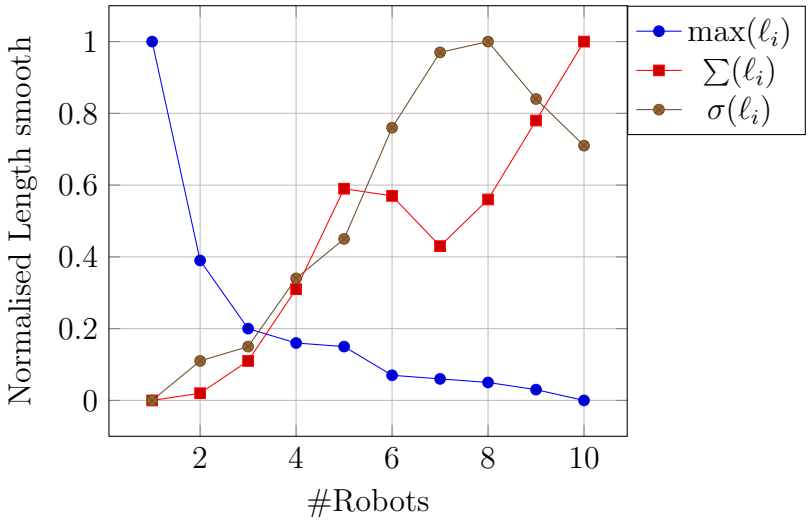

Figure 5.11: Performance indexes increasing the num. of robots, 15x15 grid using NC

In figure 5.12 is presented a comparison plot for the algorithm execution time with an increasing size of the maps. The longest execution time has been recorded for the computation of the path for one robot in a 40 by 40 free map, for a value of 7×10^2 seconds, roughly 12 minutes.

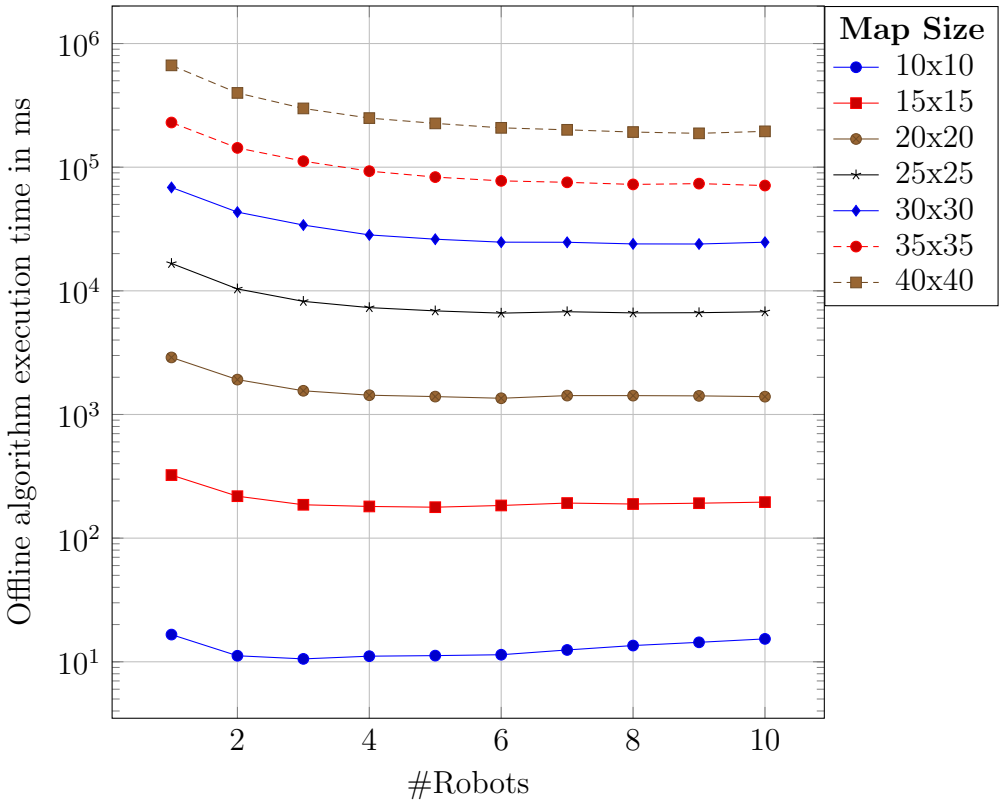

Figure 5.12: VRP-FloydWarshall: Different execution times for growing map sizes

5.3 Conclusions

We have seen how for the multi-robot coverage problem the elementary Node Counting algorithm resulted to be one of the most efficient solutions, with final paths lengths even comparable with the VRP Greedy ones. The most interesting results being the one of the "vertex oriented" algorithms Node Counting and LRTA*, the latter suffering for the intrinsic inefficient mechanism implemented for the update rule in the multi-robot case (section 3.2). "Edge oriented" algorithm as Edge Counting and PatrolGRAPH* do not seem to fit the needs of the coverage problem as they always end up in longer paths with respected to the vertex oriented algorithms. The VRP algorithm seems anyway to achieve the overall best results in very large obstacle-free maps, which can be the case for high altitude flight. An undeniable advantage of the VRP algorithm also is that the solution is almost instant giving us an idea of the total coverage time, a very useful information in SAR applications.

Chapter 6

Experiments With Real Multi-copters

As a final step the algorithms were tested using two high-end quadcopters available in the laboratory. In particular the multi-copters are produced by Ascending Technologies and are:

- Asctec Firefly (hexacopter)
- Asctec Pelican (quadcopter)

Figure 6.1: Asctec Firefly

Figure 6.2: Asctec Pelican

They both mount an on-board computer which runs Linux and Wi-Fi adapter through which is performed all the communication. For the robot dynamic control system the ETHNOS [21] real-time software framework is used. The ETHNOS environment is composed of a dedicated distributed real-time operating system (developed as an extension to Linux), from which the overall environment takes its name, with a dedicated network protocol designed for both the single robot and the multi-robot environment, specifically designed for noisy wireless communication. Using a dedicated ROS/ETHNOS interface the communication between the path planner and the controller is made possible.

6.1 Multi-rotor model and parameters tuning

During the experiments "on the field" the main part was spent in the fine tuning of the quadcopters controllers. In particular by adjusting the PID parameters that control the propellers thrust in reaching a given target. In this section we will give a general idea of the control scheme and how this applies to the quadcopter dynamics.

To describe the ideal dynamics of a multi-rotor aircraft we can use the following mathematical model [2]:

$$\begin{cases} \ddot{\xi} = \frac{1}{m} u R e_3 - g e_3 \\ M(\eta)\ddot{\eta} + C(\eta, \dot{\eta})\dot{\eta} = \Psi(\eta)^T \tau \end{cases} \tag{6.1}$$

where m, ξ and η are the aircraft mass, position and orientation respectively. (u, τ) are the applied thrust and torque vector, R and Ψ are the rotation matrix and Euler matrix respectively.
For autonomous multi-rotors it is common to separate the flight control problem into an "inner-loop" that controls attitude and an "outer-loop" that controls the translational trajectory of the aircraft. After transforming the original system into one that describes the position dynamics [13] we can finally the PID controller as follows:

$$\mu_x = -K_{P_x}(x - x_d) - K_{I_x}\int (x - x_d)dt - K_{D_x}(v_x - v_x d) \tag{6.2}$$

And the same goes for y and z. This is the so called outer-loop position control. μ is an intermediate control vector and after evaluating μ_x, μ_y and μ_z we can calculate the desired thrust u, roll angle ϕ_d and pitch angle θ_d that constitutes the input parameters for the inner-loop , the *attitude controller*:

$$\begin{cases} u = m\sqrt{{\mu_x}^2 + {\mu_y}^2 + (\mu_z + g)^2} \\ \phi_d = sin^{-1}\left(m\dfrac{\mu_x sin\psi_d - \mu_y cos\psi_d}{u}\right) \\ \theta_d = tan^{-1}\left(\dfrac{\mu_x cos\psi_d - \mu_y sin\psi_d}{\mu_x + g}\right) \end{cases} \tag{6.3}$$

where g is the gravity acceleration that has to be compensated to fly and ψ_d is the yaw angle which represent the aircraft heading direction. In general ψ_d can be set to zero, as the orientation of the multi-rotor is not relevant unless it mounts some front directional camera. The attitude controller will then care of transforming these values into input controls for the aircraft's propellers. The standard PID controller can be represented using a block diagram as in figure 6.3.

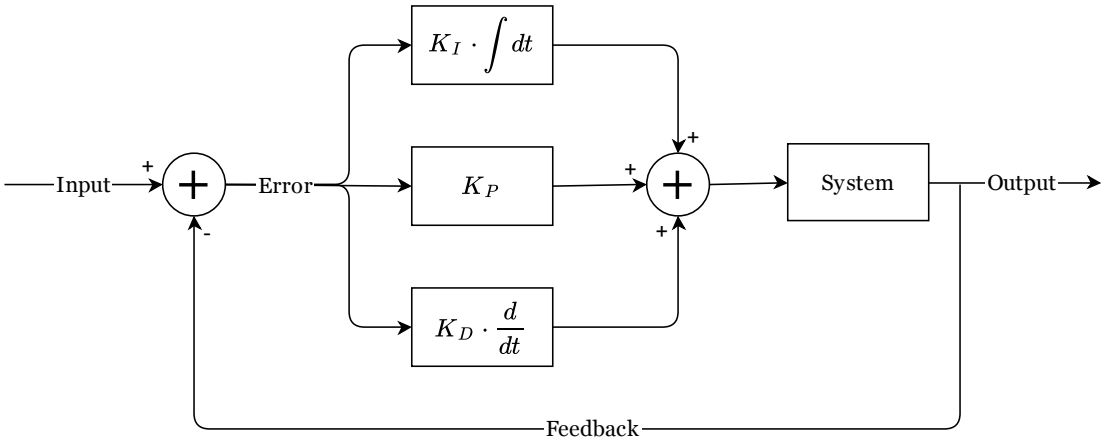

Figure 6.3: A general PID controller block diagram

To have any kind of control over the quadcopter or multi-copter, we need to be able to measure the quadcopter sensor output (for example the pitch angle), so we can estimate the error (how far we are from the the desired pitch angle, e.g. horizontal, 0 degree). We can then apply the 3 proportional (P), integrative (I) and derivative (D) control algorithms to the error, to get the next outputs for the motors aiming to correct the error.

The three parameters that we can adjust to improve better quadcopter stability are the K_P, K_I, K_D gains showed in figure 6.3. Each gain coefficent basically would change the importance and influence of each algorithm to the output. Let's now look at what are the effects of these parameters to the stability of a quadcopter.

Since we want to control the quadcopter's (x, y, z) position in space, the parameters will be described in terms of how they affect the process of keeping the desired position.

The proportional gain coefficient K_P determines how fast the quadcopter will reach the desired position. The higher the coefficient, the higher the multi-copter will be sensitive and reactive to position change. If it is too low, the quadcopter will appear sluggish and will be harder to keep steady. If the P gain is too high the multi-copter may start to oscillate with a high frequency and overshoot the target repeatedly. The integral gain coefficient K_I can increase the precision of the final position since it will eliminate the so called *steady-state error*. This term is especially useful with irregular wind, and ground effect (turbulence from motors). However, when the K_I value gets too high your quadcopter might begin to have slow reaction and a decrease effect of the proportional gain as consequence, it will also start to oscillate like having high K_P gain, but with a lower frequency. The derivative gain coefficient K_D will impact what the other two gain cannot control

that is reduce the overshoot and the settling time. It will also work like a damping factor against the previous two, to prevent excessive oscillation in a situation of high K_P and K_I. It This parameter is extremely delicate since it depends on the rate of change of the error and in certain cases it may amplify the effects of the previous gains.

Since finding the exact parameters of the mathematical model of each quadcopter can be extremely difficult the most common way to tune this parameters is by trial and error. In the following paragraph there are a couple rules of thumb. To tune quadcopter K_P gain the procedure is to slowly increase it until the quadcopter reaches the position without producing big oscillations. For the K_I gain, again start low and increase slowly, paying attention to how long does it take to stop and stabilize. You want to get to a point where it stabilize very quickly while not wandering around for too long. To get a reliable I-value a test under windy condition may be useful. For K_D gain, it can get into a complicated interaction with K_P and K_I values. When using D gain, you need to go back and fine tune P and I to keep the plant well stabilized.

6.2 Results

Here we present the results for the Node Counting algorithm

Chapter 7

Additional details

7.1 About the code

In this section a couple of technicalites are described for who is interested in deepening the knowledge of the software implementation. There is somehow a logic beyond the name of the source files: all the files that start with the *quad* prefix are controlling the quadcopters in the simulator via ROS and are consequently executable files. To plan the paths the executables make use of libraries implemented in the relative classes. For example the node counting algorithm is implemented in the `NodeCounting.cpp` class, used by `quadNodeCount.cpp` to control the movements of the quadcopters in the simulator. All the code was finally compiled with the gcc optimization level "-O3" enabled, by setting in the `CMakeList.txt` the *release* build type. The code is designed to be fairly flexible making large use of input arguments in order easily run the simulation using different parameters without having to recompile the code each time. The input parameters for the online and offline algorithms (`quad*.cpp` files) are slightly different due do the different nature of the two, and are listed in table 7.1.

argv[·]	**Corresponding Parameter**		
	Online robot controller	Offline robot controller	Offline kernelNode
1	Robot ID#	Robot ID#	Input map
2	Input map	Height of flight (z)	# of robots
3	Height of flight (z)	Control Mode	
4	Control Mode		
5	Starting vertex		
6	Min visits per vertex		

Table 7.1: Input arguments for the coverage algorithms

As we can see in the offline case the parameters are spread between the controller and the kernelNode that generate the paths. The *Control Mode* argument is used to change the waypoint planning algorithm for switching between the simulator and the real multi-copters control.

7.2 Open Licensing

> "In open source, we feel strongly that to really do something well, you have to get a lot of people involved." (L. Torvalds)

Following the attitude which inspired Torvalds, and the *not reinventing the wheel* paradigm, all the code developed in this thesis is released with open source licenses such as MIT License and BSD License. Mixing this two licenses is possible since they are both "GPL-compatible". For more info visit OSI: Licenses and License Compatibility. The source files can be found on GitHub in the VRepRosQuadSwarm repository.

The same attitude has been followed for the software tools used: Ubuntu, ROS, Coppelia Robotics VREP Simulator and LibreOffice are all open source projects. The 3D models used to create the simulation environment are free models, downloaded at TF3DM. As you probably have already noticed this thesis is written in LaTeX, also distributed under a free software license.

Appendix A

First iterations of the online coverage algorithms

To observe the behaviour of the algorithms we will look at the evolution of the *Count Map*, that is the matrix that contains the visits count for all the vertices in the graph. The map used for this purposes is 4x4 obstacle free grid-map. For simplicity let's imagine to sample the time every time we reach a new vertex (a cell of the map). To be able to look at a fairly wide time window, for all the algorithms are presented the count maps for t=0, t=3, t=6, t=9.

In the pictures we will consider the time instant when the robot has just arrived on the current cell but not yet made is decision on the next move. The cell in which the robot is currently on is highlighted in grey.

A.1 Node Counting

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Fig. A.1: t=0

1	1	1	0
0	0	1	0
0	0	0	0
0	0	0	0

Fig. A.2: t=3

1	1	1	0
0	0	1	0
1	1	1	0
0	0	0	0

Fig. A.3: t=6

1	1	1	0
0	0	1	0
1	1	1	0
1	1	1	0

Fig. A.4: t=9

A.2 Learning Real-Time A*

For the Learning Real Time A* in addition to the visits count map is presented the *LRTA-count* one, following the LRTA update rule as in algorithm 3.

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Fig. A.5: t=0

1	1	1	0
0	0	1	0
0	0	0	0
0	0	0	0

Fig. A.6: t=3

1	1	1	1
0	0	1	2
0	0	0	0
0	0	0	0

Fig. A.7: t=6

1	1	1	1
0	0	1	2
0	0	0	1
0	0	1	1

Fig. A.8: t=9

Fig. A.9: Visit count map

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Fig. A.10: t=0

1	1	1	0
0	0	0	0
0	0	0	0
0	0	0	0

Fig. A.11: t=3

1	1	1	2
0	0	1	1
0	0	0	0
0	0	0	0

Fig. A.12: t=6

1	1	1	2
0	0	1	1
0	0	0	1
0	0	0	1

Fig. A.13: t=9

Fig. A.14: LRTA-count map

A.3 Edge Counting

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Fig. A.15: t=0

1	1	1	0
0	0	1	0
0	0	0	0
0	0	0	0

Fig. A.16: t=3

1	1	1	0
0	0	1	0
0	0	1	0
0	0	1	1

Fig. A.17: t=6

1	1	1	0
0	0	1	0
0	1	2	0
0	0	2	1

Fig. A.18: t=9

A.4 PatrolGRAPH*

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Fig. A.19: t=0

2	1	0	0
1	0	0	0
0	0	0	0
0	0	0	0

Fig. A.20: t=3

3	2	0	0
1	1	0	0
0	0	0	0
0	0	0	0

Fig. A.21: t=6

3	3	1	1
1	1	0	0
0	0	0	0
0	0	0	0

Fig. A.22: t=9

Bibliography

[1] Esther M. Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1 – 18, 2006.

[2] S. Azrad, F. Kendoul, and K. Nonami. Visual Servoing of Quadrotor Micro-Air Vehicle Using Color-Based Tracking Algorithm. *Journal of System Design and Dynamics*, 4:255–268, 2010.

[3] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part 1: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[4] K.L. Choy Canhong Lin, G.T.S. Ho, S.H. Chung, and H.Y. Lam. Survey of green vehicle routing problem: Past and future trends. *Expert Systems with Applications*, 41:1118–1138, 2013.

[5] Giorgio Cannata and A Sgorbissa. A minimalist algorithm for multirobot continuous coverage. *Robotics, IEEE Transactions on*, 27(2):297–312, April 2011.

[6] Stephen Lai Chee Chong. Am 78 coverage strategies. Technical report, 2004.

[7] H. Choset, E. Acar, AA Rizzi, and J. Luntz. Exact cellular decompositions in terms of critical points of morse functions. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3, pages 2270–2277 vol.3, 2000.

[8] Howie Choset. Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots*, 9(3):247–253, 2000.

[9] Andrew Coburn et al. Death tolls in earthquakes. *The Royal Academy of Engineering*, 1994.

[10] M.L. Cummings, J.P. How, A Whitten, and O. Toupet. The impact of human-automation collaboration in decentralized multiple unmanned vehicle control. *Proceedings of the IEEE*, 100(3):660–671, March 2012.

[11] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[12] Toru Ishida. Real-time search for autonomous agents and multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 1(2):139–167, October 1998.

[13] Farid Kendoul, Isabelle Fantoni, and Rogelio Lozano. Asymptotic Stability of Hierarchical Inner-Outer Loop-Based Flight Controllers. In *17th IFAC World Congress*, page 0, Seoul, South Korea, July 2008.

[14] S. Koenig and R.G. Simmons. Unsupervised learning of probabilistic models for robot navigation. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2301–2308 vol.3, Apr 1996.

[15] Steve Koenig, Boleslaw Szymanski, and Yaxin Liu. Efficient and inefficient coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31:41–76, 2001.

[16] Sven Koenig and Reid G. Simmons. Easy and hard testbeds for real-time search algorithms. In *In Proceedings of the National Conference on Artificial Intelligence*, pages 279–285, 1996.

[17] Oscar Liang. Quadcopter pid explained and tuning, 2013. [Online at http://blog.oscarliang.net/quadcopter-pid-explained-tuning/; accessed 3-August-2014].

[18] Raphael Mannadiar and Ioannis M. Rekleitis. Optimal coverage of a known arbitrary environment. In *ICRA*, pages 5525–5530. IEEE, 2010.

[19] Navid Nourani-Vatani. *Coverage Algorithms*. PhD thesis, 1Ørsted - DTU, Automation, Technical University of Denmark, December 2006.

[20] S. Ntafos. Watchman routes under limited visibility. *Comput. Geom. Theory Appl.*, 1(3):149–170, 1992.

[21] Maurizio Piaggio, Antonio Sgorbissa, and Renato Zaccaria. Programming real time distributed multiple robotic systems. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *Lecture Notes in Computer Science*, pages 412–423. Springer Berlin Heidelberg, 2000.

[22] A. Pirzadeh and W. Snyder. A unified solution to coverage and search in explored and unexplored terrains using indirect control. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 2113–2119 vol.3, May 1990.

[23] A Renzaglia, L. Doitsidis, A Martinelli, and E.B. Kosmatopoulos. Cognitive-based adaptive control for cooperative multi-robot coverage. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3314–3320, Oct 2010.

[24] M. Schwager, Brian J. Julian, and D. Rus. Optimal coverage for multiple hovering robots with downward facing cameras. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 3515–3522, May 2009.

[25] Stanford University. Motion planning in robotics, 1999. [Online at http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html; accessed 5-08-2014].

[26] Wikipedia. Vehicle routing problem, 2014. [Online at http://en.wikipedia.org/wiki/Vehicle_routing_problem; accessed 3-08-2014].