Coverage Algorithms for Search And Rescue with UAV Drones

C. Nattero*, C. T. Recchiuto*, A. Sgorbissa* and F. Wanderlingh*

Abstract— **In order to organize effective rescue missions a prompt intervention is essential, and to this extent UAV drones are a promising technology to the end of exploring an area and gathering information before first responders come into play. In this article we investigate the development of team of drones (multicopters) implementing autonomous coverage strategies for Search&Rescue after earthquakes. In particular we survey, analyse and compare the performance of a subset of real-time multi-robot coverage algorithms in the literature, in order to classify their performance in terms of the required coverage time and, indirectly, the energy required to perform a mission. The algorithms were implemented using a distributed software framework and tested in a 3D simulation environment. We support our analysis presenting a wide set of experimental results tested in different scenarios. Finally we present some preliminary test using real robots to perform the coverage.**

I. Introduction

Autonomous robots can play a crucial role in emergency scenarios in which intervention is impossible or dangerous for humans, e.g., during natural hazards such as earthquakes, floods, or fires. In these scenarios, protocols for planning and management of emergencies usually require that human operations must be delayed until the conditions are sufficiently safe. Then, the availability of autonomous systems can be of the uttermost importance in order to gather as much information as possible before human intervention, which will support first responders involved in Search&Rescue operations in a later phase.

The work described in this article has been performed during the PRISMA project[1], which focusses on the development and deployment of robots and autonomous systems able to operate in emergency scenarios, with a specific reference to monitoring, pre-operative management, and real-time intervention. Specifically, the article describes the work that has been done concerning the development of team of drones (multicopters) implementing autonomous coverage strategies for Search&Rescue after earthquakes.

Flying robots are currently receiving a great interest from the whole scientific community as well as the market, as testified by the huge number of recent scientific articles in the field[2] as well as commercial products available. In spite of the new opportunities offered for outdoor monitoring applications, flying robots still have many limitations. The biggest limitation is probably energetic autonomy, since most commercially available products has batteries whose duration ranges between ten and twenty minutes, which is insufficient for monitoring large scale areas. Other limitations concerns such issues as wireless communication range, fault tolerance, computational power. Under these conditions, and by constraining the analysis to autonomous coverage, the availability of coordination algorithms with minimal requirements in terms of i) energetic autonomy ii) communication range iii) computational load is a *condicio sine qua non* for the deployment of flying robots for Search&Rescue applications.

The main contribution of the article is to survey, analyse and compare the performance of a subset of multi-robot coverage algorithm in the literature, i.e., real-time search algorithms. Algorithms belonging to this class do not plan the route of all robots in advance before operations, but decide the behaviour of each robot (i.e., the next area to be visited) in real-time depending on the observed behaviour of its team mates (i.e., the areas that have been visited up to the current time). Having very low requirements in terms of communication and computational power, real-time coverage algorithms turn out to be particularly promising to meet the requirements ii) and iii) of Search&Rescue with flying robots: however, their ability to produce short paths meeting the energetic requirement i) must be evaluated and quantitatively compared.

Section II surveys related work. Section III describes the algorithms selected for comparison in a common formalism. Section IV compares the selected algorithms in simulation. Section V discusses the implementation that we have performed on real robots as well as preliminary experimental results, and Section VI presents conclusions.

II. Related Work

Most approaches to coverage, either single or *multi–robot*, are based on *space decomposition*. For example, [1] assumes a single robot equipped with sensors or tools such that, in a given position, the robot can cover a square area, and imposes a *simple grid* approximation (i.e., a grid without internal holes) where each cell has dimensions corresponding to the coverage capabilities of the robot in one step. Then the grid is subdivided into rectangular sub–grids that can be covered optimally, and a Traveling Salesman Problem (TSP) is solved to compute a path that visits all sub–grids. In case of many robots [2] the problem can be formalized as a Multiple Traveling Salesman Problem – MTSP. These ideas are extended in [3] by considering *non–simple grids* which do not possess *local cut nodes* (i.e., nodes whose removal

*C. Nattero, C. T. Recchiuto, A. Sgorbissa and F. Wanderlingh are with the DIBRIS department, University of Genova, 16145, Italy; e-mails: cristiano.nattero@unige.it, carmine.recchiuto@gmail.com, antonio.sgorbissa@unige.it, fwanderlingh@gmail.com.

[1]Funded by the Italian Ministry in the context of the National Operative Program (PON) 2007–2013.

[2]In the last years, most top-ranked robotic conferences such as ICRA and IROS have one or more Sessions dedicated to flying robots.

locally disconnects the graph induced by the grid). In [4] a complete coverage algorithm is described which is based on a new type of exact cellular decomposition method, termed the *boustrophedon cellular decomposition* (i.e., based on back and forth ox–like motions). In this approach, extended to multi–robot coverage in [5], cells are computed in such a way as to guarantee that a robot can easily plan a boustrophedon motion in each cell. Similar approaches have been proposed, among the others, in [6], [7], [8], [9].

Differently from the previous approaches, which mostly rely on the idea that the work–area is decomposed into subregions and that each robot is assigned a subregion (or set of subregions) to cover, *Spanning Tree Coverage* (*STC* in short) envisions a situation in which all robots periodically cover the whole environment with the purpose of guaranteeing that all areas are visited with *uniform frequency*. *STC* for a single robot has been first proposed in [10], [11], again by assuming a robot equipped with an ideal sweeping tool, and by imposing a tool–based grid approximation. The approach relies on the idea of building the minimum spanning tree over the grid, and moving along the Hamiltonian cycle that follows the tree around. The authors show that each cell in the grid is visited repeatedly at the same frequency. These ideas have been extended to *Multi–Robot Spanning Tree Coverage* (*MSTC* in short) in [12]: after computing the Hamiltonian cycle, robots are positioned uniformly along the path, and are instructed to walk along the cycle in equidistant relative positions. According to [13], computing *MSTC* trajectories in order to minimize the coverage time is a NP–complete problem; to deal with this limitation, the article presents *Multi-Robot Forest Coverage* (*MFC* in short), a polynomial–time multi–robot coverage heuristic technique providing an improved solution. In [14] an on–line variant of *MSTC* is proposed, which assumes that the robots do not have a priori knowledge of the work–area and proves to be complete and robust in presence of noisy sensor data. In [15] a different method is described to generate Hamiltonian exploration paths for multiple mobile robots in a restricted working environment.

All of the approaches above have two major limitations when considering the constrains put by the reference scenario, i.e., envisaging teams of flying robots for Search&Rescue operations. First, computing a complete route for each robot under optimality criteria is computationally expensive. Second, if robots are assigned a route in advance before operations, the system necessarily exhibits low tolerance to failures and puts severe constraints in terms of inter robot communication and computational load. As an example, consider the situation that a robot is no more able to communicate its progress to team mates, or it must unexpectedly come back home for battery recharging: in this case re-planning the route of all robots would be required, with an additional cost and delay in operations. As another example, consider the even worse situation that a robot is no more able to progress without being aware of its fault: in this case, complete coverage cannot be more guaranteed.

Algorithms that do not suffer of the problems above exist: they belong to the domain of the so–called *real–time search* or *ant–like* methods, which have been designed to provide a general solution to complex search problems, but have been exploited for robot coverage and exploration as well. The simpler of these methods is perhaps *Random Walk*. Assume that the environment is modeled as an oriented graph[3]: when the robot is in a vertex, it selects one of the departing edges randomly with uniform probability, which guarantees complete coverage in a statistical sense as the exploration time tends to infinite. *Edge Counting* [16] is a deterministic variant of this idea in which a robot, in a given vertex, counts the number of times that each departing edge has been selected, and chooses different edges in circular order in subsequent visits. *Node Count* [17] exhibits an improved behaviour by relying on the idea of associating a value with each vertex of the graph, which counts how often each vertex has been already visited so far. When a robot enters a vertex, it increases the value of the vertex by one; next, it moves to the adjacent vertex which has received less visits up to present time. Variants of this simple idea exist (e.g., [18], [19], [17]): in particular, *Learning Real–Time A** [17] requires a look–ahead of one edge traversal, but guarantees better performance [20]. *Real–time search* or *ant–like* methods especially crafted for robotic applications have been proposed. In [21], [22], [23], a graph exploration strategy similar to *Node Count* is described which relies on the physical deployment of *pebbles* which the robot can drop or collect to recognize already visited vertices. In [24], [25] a family of ant–like approaches is proposed in which robots are able to cover the floor of an un–mapped building by physically leaving pheromone–like chemical odour traces that evaporate with time. In [26] *PatrolGRAPH** has been presented, which allows to deal explicitly with the problem of Multi Robot Controlled Frequency Coverage, i.e., when different vertices of the graph must be visited with different frequencies (e.g., depending on some priority criteria).

Existing *Real–time search* and *ant–like* approaches can be easily extended to *multi–robot* systems. One of the characteristics of this class of approach is that – in line of principle – they do not require inter-robot communication at all, given that some implicit communication mechanism "through the environment" is available (i.e., taking inspiration from the usage of pheromone in ants and termites or similar stigmergic mechanisms). In practice, the implementation of a real sensor that reliably evaluates the strength of smell of an evaporating chemical trace turns out to be technically very challenging, and the same can be said for a servomechanism which deploys and collects pebbles (not to mention the difficulty of emulating stigmergy with flying robots). Therefore, these processes are usually emulated through a global memory that is shared among all robots, which ultimately requires some sort of inter-robot communication. In spite of this, *Real–time search* and *ant–like* approaches exhibit all the required characteristics of the reference scenario: a robot that is no

[3]A grid can be mapped onto an oriented graph simply by doubling all edges that connect neighbouring cells, and by assigning them opposite directions.

more able to progress or to communicate its progresses does not produce a catastrophic effect, since all decisions are taken in real-time with a very low computational cost.

III. Coverage Algorithms

In this Section we formally define the coverage problem, and we describe a subset of the available coverage algorithms in the literature, namely: *Node Count*, *LRTA**, *Edge Counting* and *PatrolGRAPH**. All of the algorithms above can be classified as real-time search algorithms: however *PatrolGRAPH** requires, as it will be shown, an additional off-line phase prior to operations. Such off-line phase is not targeted at planning robot routes, and therefore it does not invalidate the discussion above.

A. Problem Statement

The coverage problem consists in finding a decision procedure which allows a team of robots to navigate in a workspace modelled as a navigation graph. Given that:

- G_N is a connected, non–oriented graph of arbitrary order, possibly containing cycles, which represents the topology of the free space, referred to as the *navigation graph*. 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.
- $S = \{s_i\}$ denotes the finite set of N vertices in $\hat{G}_N$. Each vertex s_i is associated with a location in the workspace.
- $A_i = \{a_{ij}\} \neq 0$ is the finite, non-empty set of directed edges that leave vertex $s_i \in S$. Each edge a_{ij} is defined through a pair of indices (i, j), which respectively identify the corresponding start and end vertices. $|A_i|$ is the dimension of the set, i.e., the number of edges departing from s_i.
- $R = \{r_i\}$ is a set of M robots. Robots are allowed to move in the workspace from s_i to s_j in $\hat{G}_N$ only if $a_{ij} \in A_i$, i.e., if the two vertices are adjacent.

The following objective must be achieved: the *M* robots must guarantee the coverage of $\hat{G}_N$ ensuring that all vertices are visited a minimum number of times D (usually $D = 1$) and that all robots return to the start vertex after the task has been accomplished (coming back home is crucial when dealing with flying robots, e.g., for battery recharging).

All of the solutions for real-time coverage examined in the following assume that the M robots concurrently execute different instances of the same Algorithm 1. The algorithm requires in input the start vertex s_{start} and the number of visits D that must be ensured to each vertex. The number of visits received by each vertex $u(s_i)$ is globally shared among all robots[4], and it is initialized to 0 for all $i = 1 \dots N$.

The algorithm itself is straightforward: when a robot is located in the current vertex s, it chooses the arc a to move along (Line 3), it updates s (Line 4), it increases $u(s)$ (Line 5), and finally it physically moves until it reaches its destination and possibly perform a task (Line 6). When all vertices have been visited at least D times, the robot heads back to s_{start}. Notice that the operator $choose(s_i, Alg)$ returns one of the arcs $a_{ij} \in A_i$ departing from s_i, and produces different coverage policies depending on the specific algorithm Alg considered, whereas $succ(s_i, a_{ij})$ returns the successor vertex s_j of s_i when moving along a_{ij}.

[4]Depending on technological constraints, this can either be implemented through a global memory, or through ant-like stigmergic mechanisms in which the information is stored in the environment itself.

Algorithm 1 Navigation Algorithm

Require: $s_{start} \in S$, D, $u(s_i) = 0$, $i = 1 \dots, N$

1: $s = s_{start}$
2: **while** $\min_i(u(s_i)) < D$ **do**
3: $\quad a := choose(s, Alg)$
4: $\quad s := succ(s, a)$
5: $\quad u(s) := u(s) + 1$
6: $\quad$ Move along edge a, perform task in s
7: **end while**
8: Go back to s_{start}

Remark 1: The number of visits received by s is increased (Line 5) before the robot physically reaches s (Line 6). As it will be clearer in the following, this is aimed at avoiding that a second robot could possibly take the same decision (i.e., heading toward s) before the first one has actually reached the target, thus producing an undesired concentration of robots in the same area. This is neither required in single-robot coverage, nor in an ideal case where the time required to reach the new vertex and possibly perform a task would be negligible. However, in a real multi-robot case, it turns out that the three operations in Lines $3 - 5$ of Algorithm 1 must be atomically executed, and this can be achieved only by increasing $u(s)$ before the robot starts moving.

B. Node Count

Node count [17] adopts a simple heuristic to decide which vertex should be chosen in Line 3 of Algorithm 1: when the robot is located in s_i, it chooses the adjacent vertex $s_j = succ(s_i, a_{ij})$ for which $u(s_j)$ is minimum (i.e., the adjacent vertex that has been visited the least number of times). In case that two or more adjacent vertices meet the selection criterion above, one of them is randomly chosen by the operator $oneof(\dots)$. Remember also that, before heading to s_j, the value of $u(s_j)$ is increased by one in Line 5 of Algorithm 1, therefore making s_j less prone to be chosen by other robots in the following.

Algorithm 2 Operator $choose(s_i$, *Node Count*$)$

1: $a := oneof(\arg\min_{a \in A_i} u(succ(s_i, a)))$
2: **return** a

C. LRTA*

*LRTA** was first described in [17], and is probably the most popular real-time search method. The algorithm can be used to find a path from a start to a goal vertex, but it is used here for coverage as proposed in [20]. Additionally to the global

variables $u(s_i)$, *LRTA** requires a value $u^*(s_i)$ associated with each vertex $s_i \in S$ (globally shared among all robots). The policy implemented by the operator $choose(\ldots)$ is then similar to *Node Count*: when the robot is located in s_i, it chooses the adjacent vertex s_j for which $u^*(s_j)$ is minimum.

However, $u^*(s_i)$ does not correspond to the number of visits to s_i, and the rule to update it is different, see Line 2 of Algorithms 3: instead of being simply incremented, $u^*(s_i)$ is now updated (before leaving s_i) on the basis of the values of adjacent vertices. In a few words, the rationale behind this update rule is to achieve the following behaviour: when choosing the next vertex to be visited, the algorithm should not select the less visited one, but it should select the adjacent vertex that "is most promising" to eventually reach an unvisited vertex. The algorithm has been shown to achieve better coverage performance than *Node Count*, see [20] for a detailed discussion.

Algorithm 3 Operator $choose(s_i,$ *LRTA**$)$

1: $a := oneof(\arg\min_{a \in A_i} u^*(succ(s_i, a)))$
2: $u^*(s_i) := 1 + u^*(succ(s_i, a))$
3: **return** a

Remark 2: Differently from *Node Count*, there is not a straightforward way to increase $u^*(s_i)$ before the robot *moves towards* s_i: in fact, the update rule in Line 2 of Algorithm 3 would require to know the vertex that will be selected after s_i, a decision that has not yet been taken before reaching s_i. For this reason, differently from $u(s_i)$, the value of $u^*(s_i)$ is updated before the robot *moves away from* s_i. In single-robot coverage this has no impact, but in a multi-robot case it turns out that choosing a new vertex s_i and updating $u^*(s_i)$ is no more atomically executed, since the physical displacement of the robot occurs in between the two operations. This leads to a counter-intuitive result: $LRTA^*$, which has been shown to be more efficient that *Node Count* in single robot coverage [20], can become very inefficient in multi-robot coverage (in the worst case all robots located in a vertex s_i could choose the same successor vertex at every iteration).

D. Edge Counting

Edge Counting is based on a different principle [16]: it selects edges departing from the current vertex s_i in circular order, in order to guarantee that, every $|A_i|$ visits, all edges are chosen with the same relative frequency $1/|A_i|$. The operator $get(\ldots)$ in Line 1 of Algorithm 4 picks an edge a in the set A_i depending on the current value of a counter $switch(s_i)$ (globally shared among all robots), and Line 2 updates $switch(s_i)$ to point to the next edge.

Algorithm 4 Operator $choose(s_i,$ *Edge Counting*$)$

1: $a = get(A_i, switch(s_i))$
2: $switch(s_i) = (switch(s_i) + 1) \mod |A_i|$
3: **return** a

Remark 3: The counter $switch(s_i)$ is updated before the robot *moves away from* s_i. However the reader can easily notice that the similarities with $LRTA^*$ are only apparent, and *Edge Counting* does not suffer any negative effects in the multi-robot case: as soon as a robot has chosen the next vertex to head to, it updates the counter, which will then route the next robot departing from s_i to a different vertex.

E. PatrolGRAPH*

*PatrolGRAPH** shares some similarities with *Edge Counting*: however, it differs in that subsequent robots departing from s_i choose edges $a_{ij} \in A_i$ in such a way that the relative frequency of the choices is not uniform, but adheres to a given distribution [27].

To this end it is necessary to define: $k(a_{ij})$ as an integer variable associated with each arc a_{ij} which counts the *actual number of robots* that have chosen to proceed to s_j after leaving s_i (globally shared among all robots); p_{ij}, $j = 1 \ldots N$ as an additional property of each edge which describes the *desired ratio of robots* that, after visiting s_i, must head towards s_j (we assume that $p_{ij} = 0$ if s_j is not adjacent to s_i, i.e., if $a_{ij} \notin A_i$).

Algorithm 5 Operator $choose(s_i, PatrolGRAPH^*)$

1: $a := oneof(\arg\min_{a \in A_i} \Delta p(s_i, a))$
2: $k(a) := k(a) + 1$
3: **return** a

The operator $\Delta p(s_i, a_{ij})$ in Line 2 computes the difference between $k(a_{ij})/u(s_i)$ (i.e., the actual ratio of robots that have chosen to move to s_j after leaving s_i) and the desired ratio p_{ij}. Then, Line 1 picks the edge a which has been chosen the least number of times, compared to the desired ratio; Line 2 increases $k(a_{ij})$ by one. Algorithm 5 guarantees that, for every edge a_{ij}, the relative frequency $k(a_{ij})/u(s_i)$ tends to the desired ratio p_{ij} at steady state [27].

The algorithm is interesting since, by analysing all the values p_{ij} taken together (which define the so called "transition probability matrix"), we are allowed to make prediction about the macroscopic behaviour of the system. Let $\boldsymbol{\lambda} = \{\lambda_i\}$, with $\lambda_i = u(s_i)/\sum_{i=1}^{N} u(s_i)$, $i = 1 \ldots N$, be the distribution of visits over vertices, i.e., the number of visits received by each vertex as time passes divided by the total number of visits. Let $\boldsymbol{\lambda}^* = \{\lambda_i^*\}$ be a desired distribution of visits. It can be demonstrated that, by properly choosing the elements of the transition probability matrix, it is possible to achieve a visit distribution $\boldsymbol{\lambda}$ which approximates, in the sense of the least squares, the desired distribution $\boldsymbol{\lambda}^*$. For instance, assume that uniform coverage is required: it is possible to compute proper values for p_{ij}, $i, j = 1 \ldots N$, such that $\lambda_i = \lambda_i^* = 1/N$, $i = 1 \ldots N$.

The computation of the transition probability matrix is performed off-line in an optimization phase which precedes operations. If we choose not to perform the optimization phase and simply set the transition probability to be the same for all departing edges (i.e., $p_{ij} = 1/|A_i|$), then the following property holds: the visiting rate λ_i is not uniformly

distributed over vertices, but proportional to the number of departing edges $|A_i|$ [27]. Notice that this is exactly what happens with *Edge Counting*, which chooses all departing edges in circular order with the same relative frequency.

Remark 4: The value of $k(a_{ij})$ is updated before the robot *moves away from* s_i. However, similarly to *Edge Counting*, the reader can easily notice that *PatrolGRAPH** does not suffer any negative effects in the multi-robot case: subsequent robots departing from s_i will use the updated value of $k(a_{ij})$.

IV. Simulated experimental results

The algorithms described in the previous paragraphs have been tested in simulation to the end of comparing their performance. All tests have been performed with the robotic simulator V-REP [28], interfaced with the ROS framework [29]. Since the final objective of the algorithms is to perform complete coverage in the shortest time as possible, thus indirectly minimizing energy consumption, two main performance criteria have been chosen.

Let us define l_i as the total length of the path travelled by robot $i = 1 \dots M$. Then the performance criteria are:

- $\max l_i$, i.e., the length of the longest path among all the robots. The smaller this value, the faster the coverage will be completed and the more likely the battery charge of each robot would be sufficient for its mission;
- $\sum l_i$, i.e., the overall distance travelled by all robots. The smaller this value, the lower the overall power consumption.

In the following analysis, the standard deviation σ_l, computed over all path lengths, is shown as well, to give an idea about the contribution of each robot in terms of coverage.

Preliminary tests have been performed with a single robot, in order to validate our implementation of the algorithms by checking if the simulated results correspond to those described in the literature. Five trials with with a 4×4 grid without internal holes have been performed, varying the minimum number of visits D that each vertex must receive. Since a single robot is used, it obviously holds $\max l_i = \sum l_i$ and $\sigma_l = 0$, therefore only the performance criterion $\max l_i$ is considered. Table I shows simulation results averaged over five trials.

D	**Node Count**	**LRTA***	**Edge Counting**	**PG***
10	178	160	250	178
5	98	79	134	108
2	51	35	75	57
1	17	17	34	47

TABLE I

Values of $\max l_i$ with different algorithms by varying the number of visits D

From these results it can be seen that the LRTA* clearly performs better than the NC algorithm, which is in accordance with the results found in literature [20]. This is more evident when D (i.e. the minimum number of visits that each vertex must receive) has a higher value; when D decreases, the difference between the two approaches becomes less considerable, and the performance is equal for $D = 1$. *Edge Counting* and *PatrolGRAPH** are less efficient, but for $D > 1$ the latter completes the coverage in fewer steps and for $D = 10$ its performance becomes comparable with *Node Count*. However, the property above is guaranteed only at steady state (e.g., by considering a sufficiently high number of visits D). Under these conditions, all vertices tend to receive the same number of visits: then, the length of the total path required to visit all vertices D times tends to be optimal.

After validating the algorithms, we performed tests for multi-robot coverage. Four Sets of tests have been performed taking into account different aspects:

- Set 1: different grid sizes, from 4×4 to 8×8 grids.
- Set 2: different graph typologies. The minimum degree of a graph, $\delta(\hat{G})$, is defined as the minimum number of incident edges in a vertex. In the followings, $M0$ is the graph corresponding to a grid without holes, $M1$ is a grid with holes, but without local cut vertices (i.e., vertices whose removal locally disconnects the graph induced by the grid), $M2$ is a grid with local cut vertices, but with $\delta(\hat{G}_N) = 2$ and $M3$ is a grid with $\delta(\hat{G}_N) = 1$. Fig. 1 shows the graph typologies $M1$, $M2$ and $M3$ used for the experiments.

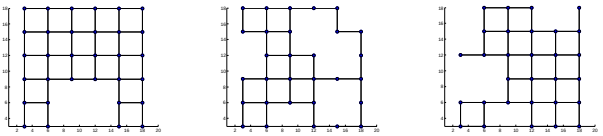

Fig. 1. Graphs $M1$ (left), $M2$ (center) and $M3$ (right), 6×6 grids

- Set 3: different number of robots, from 2 to 6.
- Set 4: different values of (D), the minimum number of visits to each vertex, from 10 to 1.

Simulations have been made by varying all relevant parameters, and the results are shown in Table II.

It can be easily seen that the *Node Count* algorithm, despite its simplicity, is the most efficient one. The results of the four Sets of simulations suggest that, by increasing the grid size and the number of robots, the performance of *LRTA** becomes clearly inferior with respect of *Node Count* (see Set 1 and Set 4). Apparently, this is in contrast with the results found in literature [20] and obtained in simulation for the single-robot case. However, the discrepancy can be explained by focusing on the update rule of *LRTA** (Remark 2). Indeed, choosing a new vertex s_i and updating $u^*(s_i)$ is not atomically executed: while in single-robot coverage this has no impact, in a multi-robot case this can lead to inefficient behaviour (in the worst case, all robots located in a vertex s_i choose the same successor vertex at every iteration).

Compared with *Node Count* and *LRTA** the values of $\max l_i$ and $\sum l_i$ related to *Edge Counting* and *PatrolGRAPH** are noticeably higher, and the performance are lower. In general, it can be observed that, when $D = 1$ (Set 1, 2, 3), *Edge Counting* tends to perform better than

(a) Varying grid size, with a $M1$ grid, 3 robots and D = 1

Set 1		**NC**			**LRTA***			**EC**			**PG***	
Grid Size	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l
4×4	7	19	0.394	7	24	0.558	18	52	0.605	25	74	0.662
6×6	20	60	0.673	19	56	0.773	69	203	1.15	65	192	0.908
8×8	37	109	0.785	43	125	0.558	142	442	1.32	150	447	1.17

(b) Varying grid typology, with a 6×6 grid, 3 robots and D = 1

Set 2		**NC**			**LRTA***			**EC**			**PG***	
Graph Type	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l
$M1$	20	58	0.605	23	68	0.762	59	175	1.04	61	182	0.836
$M2$	16	45	0.625	20	60	0.490	56	167	0.860	66	193	1.19
$M3$	19	55	0.394	16	46	0.840	46	133	0.958	69	204	1.39

(c) Varying the number of robots, with a 6×6 $M1$ grid and D = 1

Set 3		**NC**			**LRTA***			**EC**			**PG***	
N. of robots	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l
2	27	54	0.341	26	50	0.624	75	148	0.974	88	174	0.832
4	14	54	0.615	19	72	0.387	44	171	0.800	51	201	0.852
6	11	60	0.770	15	85	0.725	42	244	1.06	41	241	0.539

(d) Varying the minimum number of visits, with a 6×6 $M1$ grid and 3 robots

Set 4		**NC**			**LRTA***			**EC**			**PG***	
D	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l	$\max l_i$	$\sum l_i$	σ_l
10	114	338	1.414	154	456	1.633	205	605	4.08	180	533	3.366
5	68	202	0.577	96	286	1.00	145	430	2.38	114	339	0.816
2	30	89	0.816	31	91	0.577	75	222	0.816	70	207	0.816
1	15	44	0.816	18	53	0.816	63	188	0.816	66	198	0

TABLE II

SUMMARY TABLE OF THE RESULTS

*PatrolGRAPH**. On the opposite, when D is higher (Set 4), *PatrolGRAPH** performs better since it ensures a uniform frequency of visits to all vertices. Finally it can be noticed that, in Set 2, the results of *PatrolGRAPH** are against the trend of all the other algorithms: a smaller value of $\delta(\hat{G})$ causes longer coverage paths. This behaviour can be explained by considering that *PatrolGRAPH** requires an off-line optimization phase to compute the values of the transition probability matrix, in order to achieve a uniform distribution of visits to all vertices. Indeed, it can be demonstrated [27] that only an approximate solution in the sense of the least squares can be found to this optimization problem. By disconnecting vertices in $\hat{G}_N$ (i.e., as the transition matrix becomes more and more sparse), the number of free parameters p_{ij} decreases, and finding a solution becomes more difficult (i.e., the actual distribution of visits diverges more and more from the uniform distribution).

*Remark 5: PatrolGRAPH** has lower performance that other algorithms also because it has been designed for continuous coverage (e.g., for patrolling applications). Indeed, the algorithm owes its peculiarity to the fact that, by tuning the elements p_{ij} of the transition probability matrix, it allows for a non-uniform frequency distribution of visits to vertices. In patrolling applications, this can be crucial to allow robots to patrol different regions with different frequencies depending on priority criteria.

V. IMPLEMENTATION ON REAL ROBOTS

In the context of the PRISMA project, focussing on the development and deployment of robots and autonomous systems able to operate in emergency scenarios, preliminary tests have been performed by implementing the algorithms to control two unmanned aerial vehicles: the quadrotor Asctec Pelican and the hexarotor Asctec Firefly shown in Fig. 2. The implementation of the algorithms on actual robots was specifically aimed at testing the whole framework in a real scenario, without focusing on performance in terms of coverage (that has been analyzed in simulation). Both aerial vehicles mount an embedded computer, two ARM7 processors for on-board computation, sensors (GPS, IMU, cameras, magnetometer, barometric pressure sensor) and a Wi-Fi adapter for wireless communication.

Fig. 2. Asctec Firefly and Pelican before the flight

Since *Real-time search* approaches rely on the presence of a global memory that is shared among all robots, the coverage algorithms described in the paper are executed on an off-board controller, implemented in ROS, that, by means of the publisher/subscriber architecture, sends the target positions to the robots and periodically receives information about the current robots position. In fact, in order to execute the algorithms, the off-board controller uses the robots position to update the number of visits of each vertex (as well as all other quantities that must be globally shared) and choose the

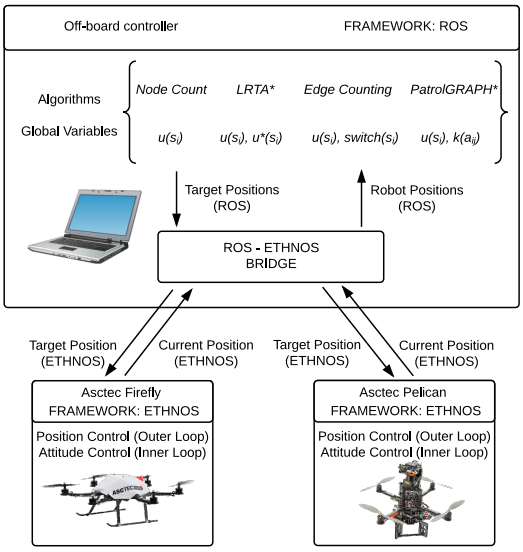

Fig. 3. Schematic description of the global system.

next target.

For the robot dynamic control and localization, performed onboard of each robot, the ETHNOS [30] multi-agent framework is used. The ETHNOS environment is composed of a distributed real-time operating system (developed as an extension of Linux) with a dedicated network protocol designed for both single robot and the multi-robot applications, specifically designed for noisy wireless communication. A ROS/ETHNOS interface has been developed to implement the communication between the robots and the off-board controller. For the navigation from the current position to the target one, two control loops are implemented on board of the robots, both based on PID controllers: an "outer-loop" that controls the trajectory of the robot using the information provided by positioning sensors and an "inner-loop", aimed at controlling the robot's attitude in space [31].

This approach based on the ROS/ETHNOS publisher/subscriber architecture has been adopted because of its robustness: e.g., if a robot is temporarily unable to communicate its position to the off-board controller, all messages (e.g., robot position, visited nodes, as well as additional quantities that must be globally shared) are still managed through ROS topics. As soon as the robot comes back on line, the messages are sent and received, thus allowing for automatic recovery.

Figures 4 and 5 show the results obtained by implementing the *Node Count* algorithm using two robots, with a simple 3×3 grid map with no obstacles, where each edge has a length of 5 meters. In particular, Figure 3 shows that, in order to avoid collisions, the two robots perform coverage at a different altitude: the vertices 4, 5, 6 in the red and blue plots are characterized by the same x and y coordinates, but a different z coordinate.

VI. Conclusions

The paper describes and surveys the performance of a subset of real-time multi-robot coverage algorithms: *Node Count*, *LRTA**, *Edge Counting* and *PatrolGRAPH**. All of

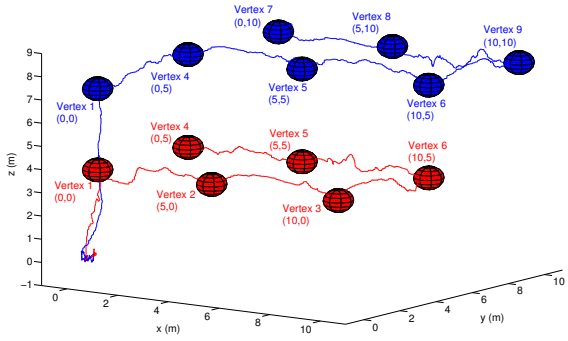

Fig. 4. 3D visual of the *Node Count* algorithm execution with the two Asctec robots. The red and the blue spheres represent the graph's vertices.

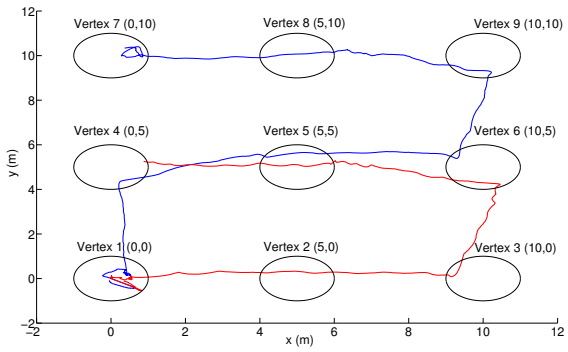

Fig. 5. Top view of the same paths of Fig. 4

these algorithms have been tested in simulation by means of the robotic simulator V-REP and the ROS framework. The algorithms have been classified in terms of the required coverage time (and, implicitly, the energy required to perform a mission). Experimental results in simulation show that, in single-robot coverage *Node Count* and *LRTA** return comparable results and outperform *Edge Counting* and *PatrolGRAPH**. However, in the multi-robot case, the simple *Node Count* algorithm turns out to be the most efficient solution: in fact, *LRTA** suffers of drawbacks due to the increased complexity of the heuristics implemented to choose the next vertex to be visited. *Edge Counting* and *PatrolGRAPH** turn out to be less efficient, both in the single and in the multi-robot case. However, when the minimum number of visits that each vertex must receive is higher, the performance of *PatrolGRAPH** become similar to *Node Count* and *LRTA*, since - at steady state - it guarantees a uniform distribution of visits. Finally, the paper describes the implementation of the system on real robots using a ROS/ETHNOS framework, as a component of the PRISMA project focussing on the development and deployment of robots and autonomous systems able to operate in emergency scenarios.

References

[1] S. Ntafos, "Watchman routes under limited visibility," *Comput. Geom. Theory Appl.*, vol. 1, no. 3, pp. 149–170, 1992.

[2] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

[3] E. Arkin, S. Fekete, and J. Mitchell, "Approximation algorithms for lawn mowing and milling," Mathematisches Institut, Universität zu Köln, Tech. Rep. 255, 1997. [Online]. Available: ftp.zpr.uni-koeln.de/pub/paper/zpr97-255.ps.gz

[4] H. Choset, "Coverage path planning: The boustrophedon cellular decomposition," in *Int. Conf. on Field and Service Robotics*, 1997.

[5] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: an algorithmic approach," *Ann. of Math. and Artif. Intell.*, vol. 52, no. 2-4, pp. 109–142, April 2008.

[6] T. Min and H. Yin, "A decentralized approach for cooperative sweeping by multiple mobile robots," in *Proc. of IEEE Int. Conf. on Intell. Robots and Systems*, 1998, pp. 380–385.

[7] W. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *Robotics and Automation, 2001. ICRA 2001. Proc. of the IEEE Int. Conf. on*, vol. 1, 2001, pp. 27–32 vol.1.

[8] Y. Guo, L. Parker, and R. Madhavan, "A multi-robot system for continuous area sweeping tasks," in *Collaborative Technologies and Systems, 2004. Proc. of the Int. Symp. on*, 2004, pp. 235–240.

[9] M. Ahmadi and P. Stone, "A multi-robot system for continuous area sweeping tasks," in *Robotics and Automation, 2006. ICRA 2006. Proc. of the IEEE Int. Conf. on*, may 2006, pp. 1724–1729.

[10] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot." *Ann. of Math. and Artif. Intell.*, vol. 31, no. 1–4, pp. 77–98, 2001.

[11] ——, "Competitive on-line coverage of grid environments by a mobile robot," *Comput. Geom. Theory Appl.*, vol. 24, no. 3, pp. 197–224, 2003.

[12] Y. Elmaliach, N. Agmon, and G. Kaminka, "Multi-robot area patrol under frequency constraints," *Robotics and Automation, 2007. ICRA 2007. Proc. of the IEEE Int. Conf. on*, pp. 385–390, April 2007.

[13] X. Zheng, S. Jain, S. Koenig, and D. Kempe, "Multi-robot forest coverage," in *Proc. of IEEE Int. Conf. on Intell. Robots and Systems*, 2005, pp. 3852–3857.

[14] N. Hazon, F. Mieli, and G. Kaminka, "Towards robust on-line multi-robot coverage," in *Robotics and Automation, 2006. ICRA'06. Proc. of the IEEE Int. Conf. on*, 2006, pp. 1710 – 1715.

[15] C. Trevai, Y. Fukazawa, H. Yuasa, J. Ota, T. Arai, and H. Asama, "Exploration path generation for multiple mobile robots using reaction-diffusion equation on a graph," *Integr. Comput.-Aided Eng.*, vol. 11, no. 3, pp. 195–212, 2004.

[16] S. Koenig and R. Simmons, "Easy and hard testbeds for real-time search algorithms," in *Proc. of the Nat. Conf. on Artif. Intell.*, 1996, pp. 279–285.

[17] R. Korf, "Real-time heuristic search," *Artif. Intell.*, vol. 42, no. 2-3, pp. 189–211, 1990.

[18] A. Pirzadeh and W. Snyder, "A unified solution to coverage and search in explored and unexplored terrains using indirect control," *Robotics and Automation, 1990. ICRA 1990. Proc of the IEEE Int. Conf. on*, pp. 2113–2119 vol.3, May 1990.

[19] T. Balch, "Avoiding the past: a simple but effective strategy for reactive navigation," *Robotics and Automation, ICRA 1993. Proc. of the IEEE Int. Conf. on*, pp. 678–685 vol.1, May 1993.

[20] S. Koenig, B. Szymanski, and Y. Liu, "Efficient and inefficient ant coverage methods," *Ann. of Math. and Artif. Intell.*, vol. 31, no. 1-4, pp. 41–76, 2001.

[21] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction," *Robotics and Automation, IEEE Trans. on*, vol. 7, no. 6, pp. 859–865, Dec 1991.

[22] M. Bender, A. Fernández, D. Ron, A. Sahai, and S. Vadhan, "The power of a pebble: exploring and mapping directed graphs," in *STOC '98: Proc. of the thirtieth ACM Symp. on Theory of computing*. New York, NY, USA: ACM, 1998, pp. 269–278.

[23] D. Xiaotie and A. Mirzaian, "Competitive robot mapping with homogeneous markers," *Robotics and Automation, IEEE Trans. on*, vol. 12, no. 4, pp. 532–542, Aug 1996.

[24] I. Wagner, M. Lindenbaum, and A. Bruckstein, "Efficiently searching a graph by a smell-oriented vertex process," *Ann. of Math. and Artif. Intell.*, vol. 24, no. 1-4, pp. 211–223, 1998.

[25] ——, "Distributed covering by ant-robots using evaporating traces," *Robotics and Automation, IEEE Trans. on*, vol. 15, no. 5, pp. 918–933, Oct 1999.

[26] M. Baglietto, G. Cannata, F. Capezio, A. Grosso, A. Sgorbissa, and R. Zaccaria, "Patrolgraph: a distributed algorithm for multi-robot patrolling," in *IAS10-The 10th International Conference on Intelligent Autonomous Systems, Baden Baden, Germany*, 2008, pp. 415–424.

[27] G. Cannata and A. Sgorbissa, "A minimalist algorithm for multirobot continuous coverage," *Robotics, IEEE Transactions on*, vol. 27, no. 2, pp. 297–312, April 2011.

[28] E. Rohmer, S. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov 2013, pp. 1321–1326.

[29] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[30] M. Piaggio, A. Sgorbissa, and R. Zaccaria, "Programming real time distributed multiple robotic systems," in *RoboCup-99: Robot Soccer World Cup III*, ser. Lecture Notes in Computer Science, M. Veloso, E. Pagello, and H. Kitano, Eds. Springer Berlin Heidelberg, 2000, vol. 1856, pp. 412–423.

[31] F. Kendoul, I. Fantoni, and R. Lozano, "Asymptotic Stability of Hierarchical Inner-Outer Loop-Based Flight Controllers," in *17th IFAC World Congress*, Seoul, South Korea, July 2008, p. 0. [Online]. Available: http://halv3-preprod.archives-ouvertes.fr/hal-00338358