Self-Organising UAVs for Wide Area Fault-tolerant Aerial Monitoring

Massimiliano De Benedetti, Fabio D'Urso, Fabrizio Messina, Giuseppe Pappalardo, Corrado Santoro
University of Catania – Dept. of Mathematics and Computer Science
Viale Andrea Doria, 6 — 95125 - Catania, ITALY
EMail: {m.debenedetti1983,fabiod}@gmail.com, {messina,pappalardo,santoro}@dmi.unict.it

***Abstract*—This paper describes an algorithm for the coordination of a flock of multirotor UAVs which cooperate in performing a wide area monitoring mission. The key characteristics of the proposed approach are the self-organising ability and the decentralisation. The flock has the basic task of covering a certain area of terrain and is able to self-organise in order to *(i)* find a coverage of the area which minimises mission time, and *(ii)* identify possible faults in one or more multirotors, taking care of performing a re-scouting of proper terrain regions in order to avoid loss of data. The algorithms to ensure flocking formation and area coverage are described. They are also validated by means of simulation approach, which is performed using an ad-hoc software tool.**

I. Introduction

The recent advent of small aerial vehicles, such as *multirotors*, made them an interesting solution also for professional applications rather than entertainment. Such multirotors have the great advantage of being easy to build; no particular skills in mechanics or avionics are required if compared with a small aeroplane which, instead, needs a particular care in wing profile, wing size and shape, tail size, position of the centre of gravity, etc. Indeed, these multirotors can be easily built by purchasing parts off-the-shelf (i.e. motors, propellers, frame, electronics) and then assembling them properly. They are basically controlled remotely by a human pilot, but they can also have the ability of flying autonomously using GPS hardware and suitable control algorithms.

Given the said characteristics, one of the most interesting applications of multirotors is the *aerial monitoring* of certain areas which cannot be accessed with other conventional vehicles: as a reference example, the authors are involved in a research project whose objective is the control of *landslips* and, since one of the monitoring strategies will be the gathering and analysis of *aerial images* in order to evaluate terrain movement, the employed solution will be indeed based on multirotors.

In addition to the cited advantages, multirotors have some drawbacks: the most critical one is the *autonomy*, which is in the order of 5 to 15 minutes (on average), not so high to make them suitable for wide areas. But another sensible problem is the *fault-tolerance*: should a aircraft fail during mission, it would be lost, together with its set of monitoring data.

To deal with the problems above, a classical solution is to employ a *set of several multirotors*, each one entailed with the objective of covering a specific portion of the monitored area; but, even if this solution solves the autonomy issue by parallelising the activities, it is not fault-tolerant, since a problem in one or more multirotors would provoke, in any case, a loss of a part of monitored data.

With these aspects in mind, the authors designed an algorithm, which is described in this paper, for coordinating a *flock of multirotors* during a wide area monitoring mission. The key characteristics of the proposed approach are the *self-organising ability* and the *decentralisation*. The flock has the basic task of covering a certain area of terrain and, to this aim, is able to self-organise in order to *(i)* find a coverage of the area which minimises mission time, and *(ii)* identify possible faults in one or more multirotors, taking care of performing a re-scouting of proper terrain regions in order to avoid loss of data. The proposed approach has been validated by using a software simulator implemented ad-hoc.

The paper is structured as follows. Section II introduces the minimum background behind this work and discusses related work. Section III introduces the use scenario, highlighting the basic characteristics and the requirements that in automated system must be met. Section IV deals with the proposed algorithms, which include flocking formation and area coverage. Section V describes the simulator implemented to test the proposed approach. Section VI reports the conclusions.

II. Background and related work

The *Area Coverage* research topic has the objective of studying solutions to cover a specific area with a certain number of robots, under a set of constraints, e.g. mission time minimisation, path optimisation and frequency cover for some point of interest. Area coverage can be classified into two main categories[6]:

- **Single Coverage:** the goal is to cover the target area until all the accessible points in the selected environment have been visited at least once.
- **Repeated Coverage:** the aim is to cover a set of points of interest in the selected environment more than once, for the entire mission.

The differences between the two aforementioned problems are closely related to the involved parameters [6]. The main parameters of Single Area Coverage are time, distance and number of visits for each point of interest. In other words, coverage must be completed while minimising the mission time, the distance covered by the robot and the frequency of visits. In Repeated Area Coverage the main parameters are the frequency of visits – to be maximised – , the length of paths

– to be minimised – and the workload distribution among the set of available robots, which has to be balanced.

The first concern of Area Coverage is the decomposition technique adopted to divide the area into fundamentals units which, in turns, are associated with a set of sub-tasks or paths. A significant approach is the Approximate Cell Decomposition [9], [11], in which the environment is divided into cells of equal size and shape, apart those cells that are partially occluded by obstacles or placed at the boundary of the area. A second approach is called Exact Cell Decomposition [10]; here the area is divided into a set of non-overlapping regions that represent the whole environment. The Boustrophedon Decomposition is a particular Exact Cell Decomposition on which the slice is a line, and the robot follows the intersection of the slice and the area to be covered with a simple back-and-forth motions. A cell in Boustrophedon decomposition is a region on which slice connectivity is constant and can be swept by a specific direction called Sweep Direction. As we discuss in the other sections of this work, our approach is similar to Boustrophedon decomposition.

Another aspect of Area Coverage is the algorithm used to cover the cells obtained from the area decomposition. As proposed in [3], the cooperation and coverage can be completely decoupled. In other words a coverage algorithm for a single robot can be extended to a cooperative setting by adding an overseer algorithm which takes incoming data from other robots and integrates it into the cellular decomposition [8]. The core of cooperation behaviours is the overseer algorithm that provides a task allocation among the robots. The computation of a specific function containing all the parameters involved in the global task [14] is thus distributed among the robots.

There are many different approaches for real time task allocation, and most of them are drawn from biology or from market-based approaches [4], [5]. Market-based approaches rely on a leader/auctioneer (a robot or an external operator) that calls an auction for each task to be assigned. The other robots will bids for each single task with an offer that represents a cost or an utility gain, and the best bidders are selected. Market-based approaches have gained popularity because they include an interesting, although implicit, fault management strategy. Indeed, whenever the communication channel between the leader (auctioneer) and one or more robots becomes unavailable, the auctioneer will call another bid for the execution of the incomplete task [1], [11]. Furthermore, if each robot broadcasts its information to the other robots, the system will be able to complete the goal even if the leader/auctioneer is affected by a fault. This additional strategy allows the remaining robots to choose another leader without loss of functionality.

The loss of communication among robots can occur in case of fault or when the distance between the robots became too large for the communication system. This second aspects becomes critical when the robots used for area coverage are UAVs or the coverage task is performed on indoor environments. In this kind of systems (UAVs fleet) a designated area must be covered with a certain level of efficiency and, at the same time, a constant connection among the UAVs must be maintained. Assuming that every UAV is equipped with a wireless transceiver (to broadcast and receive message from/to others UAV) a tree-based overlay network can be built and the area coverage tasks can be performed within network connectivity constraints [13]. Indeed, maintaining a complete knowledge of the environment requires a continuous broadcasting process among the robots that reduces the overall available bandwidth. Therefore, one of the important issues in this kind of environments (Area Coverage - UAVs fleet) is to find a trade-off between the broadcasting process and the minimum bandwidth required to perform/assign all the pending tasks.

III. System Model and Mission Requirements

We consider a system made of *(i)* a certain *physical area* that must be monitored, and *(ii)* a set of *mobile entities* with the ability of autonomously moving in the considered area and equipped with the sensors needed for the monitoring activity.

We assume that an inertial reference system exists, given in a 3D space, which includes, together with XYZ coordinates, also the *heading*. In the Earth reference system, these coordinates will correspond to *latitude*, *longitude*, *altitude* and *heading*.

On this basis, the area to be monitored can be identified by a polygonal in the XY plane of the considered reference system. Without loss of generality, we can assume that the monitored area has a *rectangular shape*[1] and therefore is identified by the coordinates of four vertices $Area = ((x1, y1), (x2, y2), (x3, y3), (x4, y4))$.

In this environment the monitoring entities live; we'll simply call them henceforward *agents*, even if, in the real scenario, they will be multirotor VTOL[2] machines. Each agent has the ability of moving in the 3D space with *four degrees of freedom*, one for each element of the coordinate system; we can define the *pose* of the i^{th} agent at time t as a 4-elements tuple:

$$pose_i(t) = (X_i, Y_i, Z_i, Head_i)$$

Each agent knows its pose by means of proper position sensor (e.g. GPS); it has also the ability of interacting with other agents by means of a *communication system* which, however, could be limited in the range (e.g. a low-power wireless system); therefore, two agents can interact to each other only if the (Euclidean) distance, in the given space, is less than the range supported by the communication system. Each agent i has also assigned a unique ID_i, which is mission-static, i.e. it does not change during a mission (for example the MAC address of the wireless card).

Each agent is also equipped with a certain *monitoring sensor*, which, given a certain Z altitude, is able to "monitor" (cover) a certain rectangle of the XY plane. As an example, if the sensor is a camera, we can determine the size of the captured pictures given the characteristics of the lens and the mission altitude. We can model this aspect by a function

$$sensor(z) = (ws_z, hs_z)$$

which takes a certain altitude z as argument and returns a tuple indicating the size of the rectangular area (the *sensor area*), in the XY plane, perceived by the sensor.

[1] If the real shape is different we can always consider the smallest rectangle which include the real area.

[2] Vertical Take-Off and Landing

In such a system, a *mission* is defined as:

$$Mission = (Area, \overline{Z}, N), N > 0$$

where:

- $Area$ is the specific area to be covered, defined as indicated above.
- $\overline{Z}$ is the altitude at which the agents must operate.
- N is the number of agents employed.

The agents must perform the mission in a complete autonomy, by cooperating with one another, and with the overall objective of

1) *to ensure the covering*, with their sensors, of the given area;
2) *to minimise* the mission time; and
3) *to avoid/minimise* the amount of overlapped data (area *over-covering* avoidance).

If we suppose that each agent i, at the end of the mission, has covered the area $SArea_i \subset Area$, the requirements above can be expressed as:

$$\cup_{i=1}^{N} SArea_i = Area \quad (1)$$

$$\cap_{i=1}^{N} SArea_i \ is \ minimal \quad (2)$$

IV. The Area Coverage Algorithm

Given the system model described so far, the area coverage approach proposed in this paper is composed of two basic parts: *(i)* a strategy to make agents fly in a certain *flocking formation*, and *(ii)* a technique to make the flock cover the given area, according to the aforementioned requirements.

A. Overlay Construction Algorithm

To support flock formation and information spreading, an overlay network among the agents is built. The aim is to let each agent know who are the other agents of the set and to make them able to exchange information about their position.

The algorithm used to construct the overlay network is based on a simple mechanism which exploits gossiping. Each agent $\overline{k}$ maintains an *agent database*, called $ADB = \{(ID_i, pose_i, HOP_i^{(\overline{k})}, TS_i)\}$ which stores, for each other agent $i \neq \overline{k}$, the agent identifier ID_i, which is also a primary key for the database[3], the current agent position $pose_i$, the number of hops $HOP_i^{(\overline{k})}$ which separates agent $\overline{k}$ to agent i, and the timestamp TS_i at which the pose information $pose_i$ has been generated.

Each agent, with a given periodicity, samples its position $pose_{\overline{k}}$, retrieves the sampling timestamp[4] $TS_{\overline{k}}$, and sends, in broadcast, the sampled information together with its ADB:

$$ADB \cup \{(ID_{\overline{k}}, pose_{\overline{k}}, 0, TS_{\overline{k}})\}$$

Since the wireless system has a limited range, the broadcast message will reach only some of the agents of the set. Each agent, on the basis of the reception of such a message, performs a comparison between its local ADB and the database received (henceforward called $RADB$); in particular, the following operations are performed:

- first the *hop count* field of each tuple of the $RADB$ is incremented by 1;
- then, for each tuple of the $RADB$, the tuple with the same ID is searched into the local ADB;
- if an associated tuple is **not found**, the $RADB$ tuple is added to the ADB;
- if the associated tuple is found, the timestamps are compared;
- if the $TS^{(RADB)} > TS^{(ADB)}$, the received information is *fresher*, so the local information in the ADB is updated;
- otherwise, the information present into ADB is kept and the received one is discarded.

This process is executed iteratively and periodically on each agent therefore, after some steps, each agent will know who are the other agents and what their "distance" is, in terms of *hop counts* and according to the transmission range of the wireless system.

Since agents always send their ADB, there could be the case in which, at a certain time instant, two different pose information, but relevant to the same agent $\overline{k}$, are travelling in the network, e.g. the one freshly generated by the agent $\overline{k}$ itself and the one previously owned by other agents and sent in their ADB: in this case, the use of timestamps permits an agent to accept the fresh information and discard the old ones, thus ensuring data consistency.

B. Ideal Flock Shape and Flocking Formation

Flock formation approaches are traditionally based on algorithms which, by running on each agent of the set, continuously perform the following steps:

1) retrieve the position of all (or a set of) the other agents;
2) evaluate a *speed* and the *direction* to follow, on the basis of some precise *rules*;
3) apply the computed speed and direction.

According to the literature [2], [7], [12], the basic rules which can drive a flock formation have to ensure the properties of *separation*, *alignment* and *cohesion*.

The first rule ($R1$, *separation*) says that if two agents are too close to each other, they must head to opposite directions. The second rule ($R2$, *alignment*) makes an agent orientate towards the average heading of the neighbour agents. The third rule ($R3$, *cohesion*) drives an agent towards a position which is the average of the positions of the neighbour agents. In addition, when the flock is used in the exploration of an area, a fourth rule is included ($R4$, *exploration*) which lets an agent drive towards the nearest unexplored zone.

All of these rules are properly weighted on the basis of their importance in forming the flock and also in order to avoid

[3] two tuples with the same ID cannot exist

[4] Timestamps are *local*, that is, they do not need a global synchronisation mechanism.

collisions and partitioning, and they must be properly adapted when a certain flock shape is desired (indeed the basic rules reported in literature often provoke flock partition and do not ensure a specific flock shape [7]). Such aspects are particularly important for our application: partition avoidance is indeed an obvious feature, while ensuring a certain flock shape means to avoid that area portions are not over-covered.

On this basis, the ideal flock shape we consider is based on a linear placement of agents along a *formation line* which is perpendicular to direction of flight, as it is shown in Figure 1. In this way, if the distance between two close agents is kept—as equal as possible—to the width of the sensor view $ws_{\overline{z}}$, the flock can monitor the area by means of slices of size $(N \cdot ws_{\overline{z}}, hs_{\overline{z}})$; since no agent is placed behind another one (in the direction of forward flight), this flock shape ensures that, during a linear flight, a certain area is not over-covered.

Fig. 1. Ideal flock shape

To make agents form the desired flock shape, our algorithm proceeds as follows.

At each step of the iteration, the agent elects a *leader*; it will be the agent with the *lowest* known ID present in the ADB[5]. Since we suppose that each agent always receives the IDs and positions of all the other agents with the overlay construction algorithm, we can assure that all agents will select the same leader. Indeed, when the overlay network construction is not in a steady-state (this transient condition occurs when the system is started and when an agent has a failure and thus abandons the flock), not all the agents may choose the same leader; however, as it will be clear in the following, this does not influence the validity of the flock formation: also flock shape will be affected by the same transient which will lead to a regime when overlay construction is a steady-state. We have to also highlight that the presence of a leader *is not* a central point of failure and does not affect the fault-tolerance of the proposed schema: should the leader fail, a new leader is elected soon and the flock can thus continue its task; as it will be detailed in Section IV-D, the transient between leader failure and new leader election does not have a significant influence on the correctness and completion of the coverage task.

The leader has the role of establishing the *formation line*, which is set as the line perpendicular to the current heading of the leader itself. The following flocking rules are applied.

The first rule, $R1$, *separation*, behaves as follows:

- The distance d_i to each other neighbour agent i is computed.
- If d_i is less than a *hard threshold* DH, no motion is applied (the UAV remains in hovering for the current iteration); this is required to avoid any possible collision.
- If d_i is less than a *soft threshold* $DS > DH$, a repulsion force is generated for the agent, whose intensity is proportional to d_i but with an heading *always parallel* to the formation line.

The *alignment* rule, $R2$, behaves by making the agent to orientate with the same heading of the neighbour which is *the nearest to the leader* (w.r.t the overlay network known). The alignment is performed by means of a yaw rotation.

The *cohesion* rule, $R3$, behaves by making the agent to move closer to the neighbour which is, also in this case, the nearest to the leader in the flock. This motion is performed by means of a translated flight. Obviously, if the agents are *too close*, rule $R1$ will ensure the right separation.

The behaviour of these rules is shown in Figure 2.

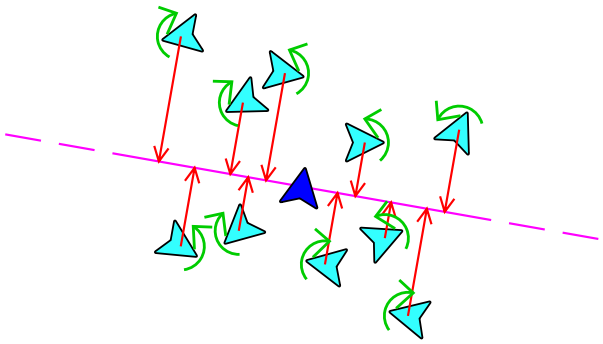

Fig. 2. Flock formation with rules R1, R2 and R3

C. Flock Driving and Area Coverage

Once the flock has been formed, its main task is to fly over the given area in order to ensure its coverage. To this aim, a *path planning* algorithm is designed. The algorithm is mainly driven by the leader and is based on the following schema:

1) All the agents spread, over the flock, the information related to the area parts already covered by each of them.
2) The leader gathers such information and computes an *optimal path*.
3) The leader starts to follow such a path thus driving the overall flock.

Such steps are executed iteratively, so that the path can be continuously adjusted on the basis of the real conditions and situations which can occur during the flight, e.g. wrong positioning of an agent due to too much wind, failure of a certain agent, etc. The key aspect of the approach lies in the way in which information on covered parts are represented, sent and processed by the agents.

As modelled in Section III, the overall area to be covered is represented by a rectangle, as well as the single portion which can be monitored by a sensor. Without loss of generality, we can perform a change of reference system and consider the rectangle area always hooked at the origin of a XY Cartesian system and with the proper width and height. In this rectangle,

[5]If the ID is not a numeric type, we suppose that a metric can be always determined in order to apply the concept of "lowest".

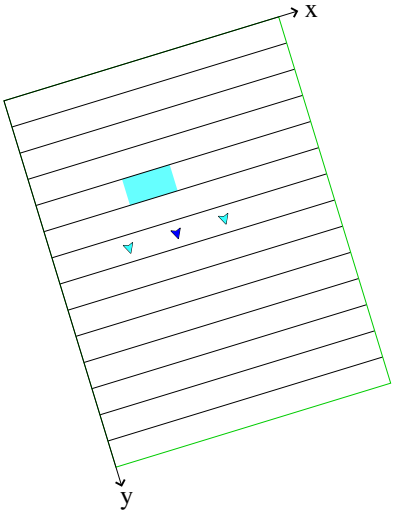

Fig. 3. Subdivision in stripes of the area to be covered

flight will be performed by keeping the *formation line* parallel to the X axis.

We subdivide the area into *stripes* by discretising the Y axis[6] (see Figure 3); the height of each stripe is equal to $hs_{\overline{Z}}$ that is the height of the rectangle covered by the sensor at the mission altitude $\overline{Z}$. We represent each stripe with a natural number, called $StripeID$ starting from 0. Since the area to cover is known at start-up time, stripe subdivision and numbering can be done before starting the mission.

Each agent, during its mission, holds and updates an *Area Parts Database* containing all the area parts already monitored. Since the parts are grouped in stripes, the database will contain a set of tuples structures as:

$$(StripeID, \{(XS_1, XE_1), (XS_2, XE_2), \ldots, (XS_n, XE_n)\})$$

Here, $StripeID$ identifies the stripe, while XS_i and XE_i indicates the stripe parts which has been already covered. The set of stripe parts is *ordered*, i.e. $XS_1 < XE_1 < XS_2 < XE_2 \ldots$ The size of each part $XE_i - XS_i$ is not forced to be equal to $ws_{\overline{Z}}$, i.e. the width of the sensor area: indeed, when several *close* sensor areas are covered, a *stripe union operation*, for such areas, is performed, as depicted in Figure 4.

The path planning approach is based on such stripe coverage data. The leader periodically starts a *distributed aggregation query* by sending, in broadcast, a data packet made of:

- Its node identifier ($LeaderID$);
- A *sequence number*, which increments each time a new query is started;
- An *agent ack field*, represented as a bitmap in which each bit indicates whether the correspondent agent has answered;
- The Area Part Database with the stripe parts already covered by the leader.

[6]We could also use X axis for discretisation, instead of Y, this choice does not affect the validity of the algorithm but only the way in which the path is then computed by the leader.

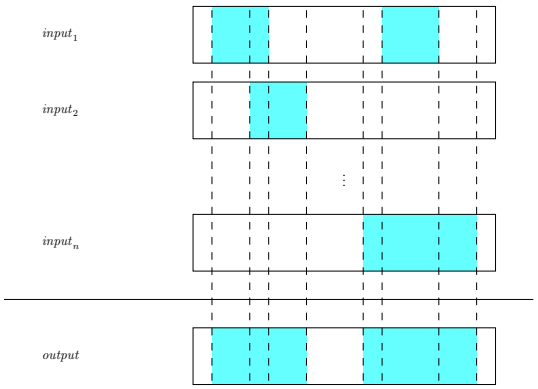

Fig. 4. Stripe Union Operation

Each agent receiving this query[7] performs the following operations:

- It first check that the $LeaderID$ correspond the its leader knowledge; if there is a mismatch, the packet is discarded (this may happen when a leader fails and a new leader is elected).
- The packet is also discarded if its sequence number is less than the last sequence number known by the agent. This is required to avoid duplicated/old data.
- The Area Part Database present in the packet is merged (through a stripe union operation) with the same database held by the agent and the result is replaced in the relevant packet field.
- The proper bit in the *agent ack field* is set.

The so-modified packet is then re-transmitted in broadcast. Re-transmission is however not performed when an agent identifies that it has *the same* knowledge of all its neighbours about the covered stripe parts (unless a new aggregated query, with a new sequence number, is started).

Sooner or later, the aggregation query packet will return to the leader, containing an updated list of stripes: if the *agent ack field* signals that *all nodes* have been answered, the list of covered stripes received by the leader can be considered completed and the next step of the path planning can be performed. To this aim, the leader computes two possible paths by following the external borders of the areas not yet covered (see Figure 5) and then chooses the one which is the closer to the flock. The chosen path is then *smoothed* (see Figure 6) and then *followed* by the leader itself: all the other agents will behave in consequence, by following the leader according to the rules $R1$, $R2$ and $R3$ illustrated above.

[7]Notice that not all the nodes receive the query, even if it is transmitted in broadcast: remember that we are considering a wireless transmission system, on each node, which could not have the adequate power to reach all the agents of the flock

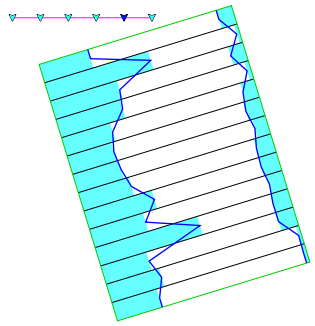

Fig. 5. Possible Paths Computed by the Leader

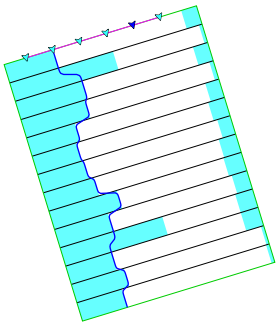

Fig. 6. Path Smoothing

In performing its flight, each agent will activate its sensor each time it will pass over a not yet monitored area part, properly updating its Area Parts Database.

As the mission proceeds, the various parts of the area to be monitored will be properly covered and the mission will end when the leader will recognise that the *whole area* has been covered: in this case, the agents can return to the base station and delivered the acquired data.

As it can be understood from the description, the proposed solution is indeed a mixture of a distributed and centralised approach. From the overall point of view, the solution is completely distributed, since knowledge of the entire system, as well as mission state, is not held by a single agent but properly spread over the various entities. Surely certain activities, like path planning, are instead centralised; indeed, this is a choice by design: in order to ensure total area coverage, flight time minimisation and over-coverage avoidance, the execution of path planning in a *centralised entity* can guarantee that an optimal solution can be found. However, such a centralised entity is not prefixed but continuously chosen during the flight on the basis of the said strategies. Therefore, as it has been introduced in Section IV-B, the use of a central entity does not have to be considered as a reliability issue, but the fault-tolerance is assured by the mechanism described below.

D. Leader Role and Fault Tolerance Aspects

When the leader fails, it stops transmitting its packets and thus the following situations will occur.

First of all no packet related to overlay construction, and coming from the (faulty) leader will transit over the network. After a certain timeout, each agent will recognise that the entry, in the ADB, relevant to the faulty leader *is no more updated*: so it can delete the entry, assuming that the leader is no more working, and elect a new leader. This process will be executed by all the agents, but using different reaction times tied to the fact that information spreading on the network, based on gossiping, is not immediate; therefore, it will be exist a *temporary interval* in which all agents *do not agree* on which node is the leader; however, this condition is not a problem: basically, agents do not directly follow the leader but their neighbours, therefore they will continue to stay in formation, naturally compensating the latency required to elect the new leader and thus continuing the mission.

A similar situation occurs when a non-leader agent fails. Also in this case, the relevant entry in the ADB will, sooner or later, become *too old* and thus discarded. The new overlay network will be built without the faulty agent so that a consistent knowledge of the new formed system is obtained, and the system itself can continue to behave correctly.

V. Simulation Results

To validate the approach proposed in this paper, a graphic software simulator has been developed, a screen-shot of which is reported in Figure 7. The simulator has been implemented in C++ using the Qt graphic libraries and allows a researcher to setup a mission, inspect its evolution and evaluate its correctness. The initial setup consists of establishing the number of agents and the area to be covered, as well as mission parameters, such as the sensor area size and the maximum range of the transmission system. After setup, the simulation can be started: first the agents self-organise in the flock and then they start the area monitoring mission. During the mission, a user can:

- dynamically add and remove agents (agent removal can be used to simulate faults);
- modify the area to be explored;
- inspect the databases of each agent;
- add/modify packet-loss percentage in order to simulate problems in the wireless system.

Simulation is time-driven: at each "time tick", a service procedure is executed for each agent, which performs all the activities described in this paper. Subsequently, the graphic display is updated by drawing the agents in their proper position and heading, together with some lines which represent the overlay network created; the area portions covered during the mission are then progressively coloured.

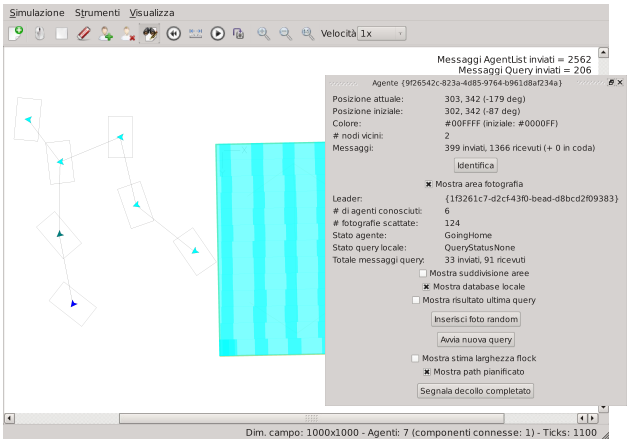

Fig. 7. Screenshot of the simulator

VI. Conclusions

This paper has described a decentralised approach to drive a set of UAVs in performing an *aerial monitoring mission* of a wide area. The approach is based on two algorithms. The first one drives agents to form a *flock* with certain given characteristics. The second one allows agents to follow a certain path which ensures the overall coverage of the area to be monitored.

The approach has been validated by means of a simulation study, performed using an ad-hoc software tool. The simulation campaign performed showed that the approach is able to drive UAVs in executing the right monitoring task, ensuring fault tolerance and area coverage, also minimising over-covering and thus mission duration.

Future work will aim at making the simulator more realistic by including the knowledge of some real physical aspects, such as the real area size, flight speed and UAV dynamics, in order to perform a proper evaluation of the real mission time. The implementation on real UAVs will be then a further step.

VII. Acknowledgements

This work is partially supported by projects PRISMA PON04a2 A/F and CLARA funded by the Italian Ministry of University.

References

[1] M. Bernardine Dias, M. Zinck, R. Zlot, and A. Stentz, "Robust multirobot coordination in dynamic environments," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 4. IEEE, 2004, pp. 3435–3442.

[2] N. Bouraqadi and A. Doniec, "Flocking-based multi-robot exploration," in *Proceedings of the 4^{th} National Conference on Control Architectures of Robots*, Toulouse, France, 2009.

[3] Z. J. Butler, A. A. Rizzi, and R. L. Hollis, "Complete distributed coverage of rectilinear environments," 2000.

[4] M. B. Dias and A. Stentz, "A market approach to multirobot coordination," DTIC Document, Tech. Rep., 2000.

[5] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[6] P. Fazli, A. Davoodi, and A. K. Mackworth, "Multi-robot repeated area coverage," *Autonomous robots*, vol. 34, no. 4, pp. 251–276, 2013.

[7] Gbor Vsrhelyi, Csaba Virgh, Gerg Somorjai, Norbert Tarcai, Tams Szrnyi, Tams Nepusz and Tams Vicsek, "Outdoor flocking and formation flight with autonomous aerial robots," in *Submitted for publication to the IEEE/RSJ International Conference on Intelligent Robots and Systems*, ser. IROS '14, Chicago, IL, USA, 2014.

[8] C. S. Kong, N. A. Peng, and I. Rekleitis, "Distributed coverage with multi-robot system," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 2423–2429.

[9] J.-C. Latombe, "Approximate cell decomposition," in *Robot Motion Planning*. Springer, 1991, pp. 248–294.

[10] ——, "Exact cell decomposition," in *Robot Motion Planning*. Springer, 1991, pp. 200–247.

[11] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: an algorithmic approach," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 109–142, 2008.

[12] C. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14^{th} Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 25–34.

[13] J. Schleich, A. Panchapakesan, G. Danoy, and P. Bouvry, "Uav fleet area coverage with network connectivity constraint," in *Proceedings of the 11th ACM international symposium on Mobility management and wireless access*. ACM, 2013, pp. 131–138.

[14] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," in *AAAI/IAAI*, 2000, pp. 852–858.