# Distributed Aerial Scanning in Mobile Ad-Hoc Networks *

Kleoni Ioannidou
Universitá di Roma *La Sapienza*, DIS
Via Ariosto, 25, 00185
Roma,Italy
ioanninu@cs.toronto.edu

Alessia Milani
Universitá di Roma *La Sapienza*, DIS
Via Ariosto, 25, 00185
Roma,Italy
milani@dis.uniroma1.it

## ABSTRACT

We formally define the aerial scanning problem. A set of unmanned aerial vehicles (UAVs) cooperate to frequently scan a given geographic area. Each UAV can only photograph a small portion of it at a time. Each UAV maintains the necessary information to create a global picture of the geographic area. This global picture consists of smaller pictures that are taken within small time of each other and also none is taken too far in the past. UAVs exchange information over a synchronous mobile ad-hoc network. For this network, we propose efficient solutions to the aerial scanning problem that tolerate a bounded number of UAV failures.

## Keywords

Mobile ad-hoc networks, UAVs, aerial scanning, distributed systems.

## 1. INTRODUCTION

Unmanned aerial vehicles (UAVs) are aerial vehicles that move without the existence of an on-board crew. UAVs are useful for dangerous or monotonous missions. In such cases, it would be risky for human pilots to fly on top of the geographic area of interest. UAVs are useful for collectively keeping track of large geographic areas.

Using UAVs for geographic aerial scanning is a challenging research topic [7]. The difficulties arise from the necessity of photographing large geographic areas, while considering the reduced cameras' field of UAVs. The collected image must be ideally constructed so that it is indistinguishable from a single large image of the same area taken at once. Moreover, complete refreshed pictures must be provided in real time in order to be useful. Several applications (e.g. convoy protection) can advantage by using UAVs to photograph areas.

*The work described in this paper was partially supported by the Italian Ministry of Education, University, and Research (MIUR) under the ISMANET project and by the European Community under Resist Network of Excellence.


AUTONOMICS 2007, 28-30 October 2007, Rome, Italy

DOI 10.4108/ICST.AUTONOMICS2007.2186

Current solutions consider the existence of a mobile ground station or a static ground station to process the images collected by each UAV independently. On the other hand, we propose a solution which does not rely on a ground base station and which is tolerant to the crashes of at most some fixed number of UAVs. For our solution, we also assume a reasonable upper bound on the frequency at which UAVs crash. Communication between UAVs is achieved by exchanging messages between wireless devices placed on the UAV. The communication service that we use provides the UAVs with inexpensive operations that allow them to broadcast messages to a small local area and receive messages previously broadcast by UAVs that are not too far away.

Each UAV can take pictures of a subarea while cooperating with the rest of the UAVs in composing such pictures to obtain a global one. This is useful for basic operations (e.g. obstacle avoidance) when a ground station does not exist. Such a scenario could appear in military operations or after some physical disaster. Our work focuses on the ability to provide global images to UAVs as frequently as possible while guaranteeing some consistency (i.e., the pictures creating the complete image must be taken within some given time). We formally define this problem, that we call the aerial scanning problem, in Section 2.

If too many UAVs are available, the aerial scanning problem can be solved relatively easily. We propose such solutions in Section 4. On the other hand, if there are not enough UAVs, then it is impossible to solve this problem as we explain in Section 4. For any number of available UAVs in between these two extreme cases, we develop an efficient solution to the aerial scanning problem and prove its correctness. Our solution is efficient for multiple reasons: first, it does not cause flooding in the network (this could have been caused if global broadcasting were used instead to distribute easily, but with large cost, local information among all UAVs); second, the moving pattern of the UAVs is determined in real time by performing exclusively local communication (i.e. local broadcasts). The fact that no flooding is caused by our solution is important because in wireless communication if too many broadcasts happen concurrently (as it would be necessary for our problem), communication can fail. This behavior can be catastrophic for applications of the aerial scanning problem. This is because it may cause the global pictures maintained by the UAVs to be too inconsistent due to lack of frequent communication. Determining the moving pattern of the UAVs in real time is important because if

some UAVs crash, it may be necessary to replace them so that all areas continue being frequently photographed. The replacing UAVs may have to travel a long distance to reach the area in need. To communicate this to distant UAVs, we could have used existing routing or geocasting algorithms (developed for mobile ad-hoc networks). Instead, we managed to solve this problem by exclusively performing local broadcasts which are less costly.

The main idea of our solution is that some UAVs are programmed to move within small subareas to photograph them continually while the remaining UAVs are moving from subarea to subarea to verify and ensure that all such subareas are being frequently photographed. These roles may change due to crashes. These subareas are small enough so that each can be photographed by a single UAV in a meaningful way (i.e., within small time so that the resulting global image that uses these images is not inconsistent). To distribute the local images among all UAVs, we design virtual mobile nodes (that we call agents) that are hosted on UAVs and that move between subareas collecting and broadcasting images. To make our solution fault tolerant we initiate multiple such agents which are never allowed to be hosted by the same UAV. In this way, even if many UAVs hosting agents crash, there will always be at least one agent in the system. Virtual agents may move together with their hosting UAV (which requires minimal communication), or move from UAV in one subarea to another UAV in an adjacent subarea. Although the latter may cost more, it is still relatively inexpensive because the subareas are small and hence, the UAVs that need to exchange information are located very close to each other.

In Section 2, we present a formal description of the aerial scanning problem. In Section 3, we describe a mobile ad-hoc model suitable for particular applications of the aerial scanning problem. In Section 4, we propose solutions of the aerial scanning problem in our mobile ad-hoc network. In Section 5, we discuss related work, and finally, in Section 6, we conclude the paper with some future work.

## 2. THE AERIAL SCANNING PROBLEM

The aerial scanning problem requires to frequently photograph a given large geographic area $A$ using a set of unmanned aerial vehicles (UAVs). Because of the reduced cameras' field, each UAV cannot take a complete picture of the area at once. But each UAV can photograph a subarea of $A$ to get its *image* when located directly on top of it. A complete picture of $A$ is a collection of *images* possibly taken at different points in time by different UAVs.

To solve the aerial scanning problem, each UAV has to maintain a collection $G$ of such frames, such that the following properties hold:

PROPERTY 2.1. *The images stored in $G$ are non-overlapping and their concatenation forms $A$.*

PROPERTY 2.2. *Each two images stored in $G$ are taken within $k$ time units of each other.*

PROPERTY 2.3. *There is finite integer $h > 0$ such that each image stored in $G$ is taken at some time during the last $h$ time units.*

Property 2.1 ensures that the images stored completely cover the geographic area $A$. Property 2.2 ensures that the images whose concatenation gives the global picture are not taken too far apart. Finally, Property 2.3 ensures that every image that is part of the global picture is not too old. The smaller is the value of $h$, the smaller is the time to have the global picture refreshed.

*Practical Applications.* Consider a military operation in which UAVs are sent to photograph a distant enemy area far away from their base station. While scanning the area and while traveling to the area, they may crash. Furthermore, when they arrive at this area they cannot contact the base station due to their distance. Another application for this problem is to deal with a physical disaster where there is no central base to coordinate movement and communication. Assume that in this area there are several help centers that can get orders from the UAVs when those are in their proximity. Scanning the area and passing the information to all UAVs allows the UAVs to have a global view which is used to direct help in a useful way. For example, as time goes by it may be discovered that some areas may have some survivors and then, help is directed towards these areas.

## 3. A MODEL FOR MOBILE AD-HOC NETWORKS

In this section, we present a model of a mobile ad-hoc network that is suitable for the particular application of aerial scanning. We consider a set of $n$ mobile nodes (each of which corresponds to a UAV) with unique identifiers which move in a 2D environment in a continuous manner with the same fixed speed. We consider a 2D environment because UAVs usually fly at the same level with small variations to avoid colliding with each other. Furthermore, in practice UAVs fly with approximately the same constant speed with the exception of taking off and landing.

Each mobile node has access to a global clock and the system is synchronous. For simplicity of presentation, the duration of a (synchronous) round is one time unit (i.e., in $[t, t+i]$, $i$ rounds have elapsed). To simplify the movement pattern that we consider, we assume that the mobile nodes move on a square grid covering the geographic area $A$ as illustrated in Figure 1. At the beginning of each round, each mobile node is placed on a point of the grid. During one round a mobile node can remain still or move from one grid point to an adjacent one. There can be more than one mobile nodes at each grid point at each time. Each node can take at most one picture of a small area (i.e. an image) at the beginning of each round. The image taken by a node placed on grid point with coordinates $(x, y)$ corresponds to the square of the geographic area with points $(x - \frac{1}{2}, y - \frac{1}{2})$, $(x + \frac{1}{2}, y - \frac{1}{2})$, $(x + \frac{1}{2}, y + \frac{1}{2})$, and $(x - \frac{1}{2}, y + \frac{1}{2})$, as illustrated in Figure 1. When a node takes a picture of the above square, we say that the node takes a picture of the grid point it is on.

Two nodes $p$ and $p'$ are *neighbours* at some time $t$, if their distance at time $t$ is no more than $r$ grid points away. We

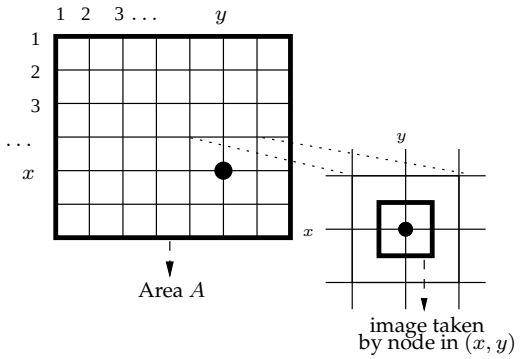


**Figure 1: Grid placed on Area $A$.**

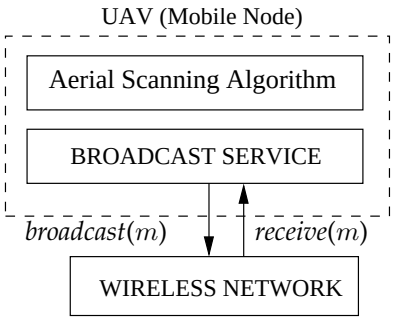


**Figure 2: System Architecture.**

assume that each node can have at most $H$ neighbours at each time. Mobile nodes can communicate with their neighbours by broadcasting messages. At the beginning of each synchronous round a node can perform receiving at most $H$ messages, broadcasting at most one message, and local computation. To perform a local broadcast of a message $x$, a node $p$ is provided with a primitive denoted *broadcast*$(x)$. If this broadcast happens at the beginning of some round $t$, then $x$ will be received at the beginning of round $t+1$ by all neighbours of $p$ at time $t$ that did not fail. Then, these nodes would generate a *receive*$(x)$ event. If two or more nodes perform broadcasts concurrently there may be interference and messages may be lost. We assume this problem to be dealt by a lower level communication layer [1]. There is no other way that messages can be lost.

*Failures.* We assume that at most $f$ nodes can fail during any execution. Nodes can fail only by crashing. An algorithm is *$f$-resilient* if it works correctly even if at most $f$ nodes crash. We also assume an upper bound on the frequency of failures. In particular, we assume that at most one failure can happen during every $F$ rounds.

# 4. ALGORITHMS FOR THE AERIAL SCANNING PROBLEM

The goal of the aerial scanning problem is to continuously deliver up-to-date images of a geographic area to all nodes of the system that do not crash. The aerial scanning problem can be solved by a service, implemented by an algorithm which runs on UAVs. As illustrated in Figure 2, on each UAV there is a process running the aerial scanning algorithm and the broadcast service we described in Section 3. We call such UAVs *(mobile) nodes*.

Each node can take an image of $k$ grid points in exactly $k$ rounds. Because of Property 2.2, every node can scan at most $k$ grid points in a useful way because otherwise the images it would take it would be taken in more than $k$ rounds apart. We conclude that it suffices for a single node to move continually in a square of no more than $k$ grid points and take images to contribute to the global picture that will be distributed among all nodes. We divide the grid in squares, called *vicinities*, of $k$ grid points each (with the exception of the last row and column which may have less). If a node is assigned to move continually within a vicinity and take pictures of its grid points, we say that it *covers* this vicinity. Assume that there are $m$ such vicinities, each identified by an integer in $[1, m]$. The *status* of a vicinity can be *uncovered*, if no node is covering it, or *covered* if at least one node is covering it. Two vicinities are *adjacent* if they share an edge.

Parameter $k$, as described in Property 2.2, must be a small time period. Otherwise, the global image that consists of submimages taken within $k$ rounds does not have practical value. Because in reality it takes some time to move from one grid point to another, then within $k$ rounds, a UAV cannot move too far away. As a result, it makes sense to assume that the vicinities are small and in particular much smaller than the areas covered by the transmission radius $r$. For the rest of our paper, we assume that $r \geq 3\sqrt{k}$. This implies that the neighbourhood of a node that is moving within the grid points of a vicinity contains all nodes in its vicinity and its adjacent vicinities.

Whether the whole geographic area can be successfully photographed by the UAVs depends on the number $n$ of the available UAVs. We consider the following cases:

If $n \geq m(f+1)$, then there can be at least $(f+1)$ nodes assigned per vicinity. Even if $f$ nodes fail each vicinity will have at least one node to cover it. The node with identifier $i$ can be located in the vicinity with identifier $i \mod m + 1$ to cover it.

If $n < m + f$, then there are not enough nodes to take images often enough of the geographic area. The reason is that if $f$ nodes fail and $n < m + f$, to ensure Property 2.2, there must exist a node that can take more than $k$ pictures within $k$ rounds, which is impossible according to our model.

Otherwise, $n \in [m+f, m(f+1))$. For this case, we describe an $f$-resilient algorithm in the next subsection.

## 4.1 An $f$-resilient algorithm for $n \in [m+f, m(f+1))$

To solve the problem of aerial scanning, first, we need to ensure that the UAVs completely cover the geographic area (i.e., to take pictures of all subareas forming this geographic area). Then, the most recently collected pictures must be propagated among all UAVs in the system to construct the complete picture of the geographic area. A complete picture must be generated by each correct (i.e., that has not failed)

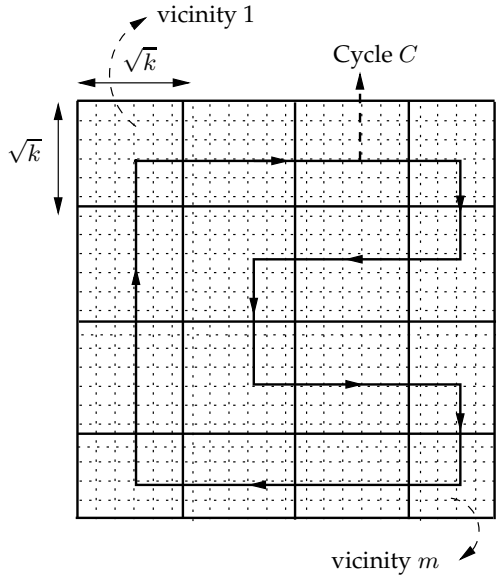


**Figure 3: Vicinity Division and Cycle $C$.**

UAV with some given frequency. Furthermore, this picture must be consistent, (i.e., it must be composed by sub-pictures taken within $k$ rounds).

*Coverage of the geographic area.* Originally, $m$ nodes are assigned, each to cover exactly one of the $m$ vicinities. We call these nodes the *covering* nodes. Each covering node moves continually from grid point to grid point in its vicinity photographing them by following a simple path. This path, ensures that each grid point of the vicinity is photographed at least once every $k$ rounds. This is possible because there are at most $k$ grid points in each vicinity and it takes one round to move from grid point to grid point and take a picture. The node keeps a history of the pictures it takes. As we will show in the proof of correctness, it suffices that it keeps the last $k + T''$ pictures it has taken, where $T'' = 2(2T + 10 + \max\{\sqrt{k}, 2m\})m$ rounds.

Since at most $f$ nodes could crash, we use the remaining (existing) nodes to ensure fault tolerance. These are at least $f$ such nodes because $n \geq m+f$. These nodes are programmed to move from vicinity to vicinity to ensure that each of them has a node that covers it. We call such nodes *vagabond* nodes. A vagabond node, moves from vicinity to vicinity according to the ordering of vicinities given by $C$. $C$ is a simple cycle that contains all vicinities. The ordering of the vicinities according to cycle $C$ ensures that two consecutive vicinities in the cycle are adjacent. An example of such cycle is illustrated in Figure 3. Each vagabond node would move following a shortest path from a vicinity to an adjacent vicinity if it did not need to perform any computation in each vicinity it is in. Otherwise, it would perform the necessary computation and then move. In the worst case, the length of this shortest path would be $\sqrt{k}$ because each vicinity is a square of at most $k$ grid points.

When an uncovered vicinity is found, the vagabond node with the smallest identifier among the ones currently in the vicinity will cover this vicinity from there and on. This node changes from being a vagabond node to being a covering node. This procedure ensures that all vicinities will be covered by some node eventually.

*Global picture delivering.* As said above, to solve the aerial scanning problem, we also need to ensure that enough information (i.e., enough images taken by the UAVs) get frequently distributed among all nodes. Because frequent updates are necessary, it is not efficient to globally broadcast the information. This would cause frequent flooding in the network.

We propose a solution where information circulates with virtual nodes, called *agents*, among vicinities. Each agent is incarnated by a mobile node at each time. We initiate $f + 1$ agents that are always located on different nodes, to tolerate $f$ failures. In this way, even if $f$ nodes with agents fail, there will always be an agent in the system performing the necessary operations. Each agent has a unique identifier from the set $\{a_1, \ldots, a_{f+1}\}$.

Each agent has to move to visit all vicinities. We choose to make the agents move from vicinity to vicinity according to the order given by cycle $C$ which is followed by the vagabond nodes.

Choosing vagabond nodes to operate agents is convenient, since agents can move from one vicinity to another one thanks to the physical movement of the hosting nodes. On the contrary, an agent incarnated by a covering node, in order to move, has to migrate to a different node.

Because of this observation, we assign as many agents as possible to vagabond nodes and the remaining agents to covering nodes. We note that there may not be enough vagabond nodes to host all agents. Furthermore, a vagabond node hosting an agent could change to being a covering node, and hence, it will never move out of the vicinity it covers.

During the execution of the algorithm, if a vagabond node without an agent moves into a vicinity whose covering node has an agent, then the agent moves from the covering node to the vagabond node and travels with it from then and on.

Next, we present our algorithm in more detail. For the following, let $T = (6 + \sqrt{k})(m - 1)$, $T' = (6 + \sqrt{k})(m - 1) + 2$, and $T'' = 2(2T + 10 + 4m + \sqrt{k})m$ rounds.

### *Data Structure*

Each node $p_i$ locally stores the following variables:

- $\texttt{status}_i$ specifies whether a node $p_i$ is covering or vagabond. In particular, a covering node has its status variable set to the identifier of the vicinity that it covers. A vagabond node has its status variable set to $\perp$ to point out that it is not associated with any vicinity.
- $\texttt{IMAGES}_i$ is an array of size $k + T''$. Each entry of IMAGES is a pair composed by a picture of a grid point and the time this picture has been taken. Node $p_i$ updates $IMAGES_i$ iff

$p_i$ is a covering node of some vicinity $v$. Thus, it contains pictures related to this vicinity. Each vagabond node also stores this local structure but it does not update it unless it becomes a covering node.
• $\texttt{GP}_i$ is an array used by $p_i$ to store the picture of the entire geographic area to be covered. In this array, we keep exactly one image per grid point.

Finally, node $p_i$ stores the following local variables which are used when it incarnates an agent:

• $\texttt{agent}_i$ is a pair (ID,COLLECTION$_{ID}$) where ID is the identifier of the agent incarnated by $p_i$ and COLLECTION$_{ID}$ is an array of $m$ arrays $IMAGES$, one for each vicinity in the grid. In particular, ID is equal to $a_j$ when node $p_i$ incarnates some agents $a_j$. Otherwise it is equal to $\perp$. $COLLECTION_{ID}[v]$ is the array of $k + T''$ entries to store images of vicinity $v$. COLLECTION$_ID$ is updated only if $p_i$ incarnates an agent. Especially, if $p_i$ is in vicinity $v$, COLLECTION$_ID$[$v$] is set equal to $IMAGES_i$.
• $\texttt{agentdone}_i$ is a boolean variable set to false if the agent has not collected yet the images of a given vicinity and true otherwise.
• $\texttt{waitAgent}_i$ is a variable used to check if $p_i$ is waiting for the transfer of an agent or not. It takes either the identifier of the agent it is waiting for, or value $\perp$ otherwise.
• $\texttt{agentmove}_i$ is a boolean variable set to true if the agent is on a covering node and it should move to the next vicinity, and false otherwise.

In the following, we describe the functions that we use in the Aerial Scanning algorithm when called by node $p_i$.

- `nextgrid(`$l, v$`)`: returns the next grid point in vicinity $v$ following the grid point $l$ according to a simple cycle containing all grid points in $v$.
- `take_picture(`$l$`)`: a mobile node invokes this function to take a picture of the grid point $l$. This call returns a pair (image, time), where time is the round at which the image it taken.
- `move_to_grid(`$l, v$`)`: notifies the mobile node that runs it to move from location $l$ to the next grid point in vicinity $v$, following a simple cycle that contains all grid points in $v$.
- `move_to_vicinity(`$v$`)`: notifies the mobile node that runs it to move from vicinity $v$ to the next vicinity according to the ordering specified by cycle $C$.
- `nextvicinity(`$v$`)`: returns the next vicinity after $v$ according to the ordering of the cycle $C$.
- `UpdateGlobalImage(INFO)`: updates the (local) variable $GP_i$ with new information stored in the variable COLLECTION of some agent, if INFO=COLLECTION, or with another global image $GP$, if INFO=$GP$. If it uses the pictures in COLLECTION, it carefully chooses a subset of them so that these pictures are not taken in more than $k$ time units apart, when possible.
- `min(b)`: given a buffer $b$ of messages, min(b) returns the minimum identifier among the identifiers of the nodes that broadcast the messages contained in $b$.

### *Aerial Scanning Algorithm*

Initially, the $m$ nodes with the greatest identifiers are assigned to cover a vicinity (line 2 of the INIT procedure in Figure 4). More precisely, node $p_i$ will cover the vicinity with identifier equal to $i - n + m$. The remaining nodes are vagabond nodes, i.e. no vicinity is assigned to them. Recall that $status_i$ denotes the vicinity covered by node $p_i$. If $status_i$ is equal to $\perp$ then node $p_i$ is a vagabond node.

```
INIT
 1  if (i ∈ [1, n − m]) then   status_i ← ⊥;
 2  else   status_i ← (i − n + m);   % p_i covers vicinity (i − n + m) %
 3  IMAGES_i ← [⊥, . . . , ⊥];
 4  if (i ≤ f + 1)
 5  then COLLECTION_{a_i} ← [[⊥, . . . , ⊥], . . . , [⊥, . . . , ⊥]];
 6       agent_i ← (a_j, COLLECTION_{a_j})
 7       agentdone_i ← false;
 8  else agent_i ← (⊥, [[⊥, . . . , ⊥], . . . , [⊥, . . . , ⊥]]);
 9       agentdone_i ← true;
10  waitAgent_i ←⊥;
11  agentmove_i ← false;
12  GP_i ← [⊥, . . . , ⊥]
```

**Figure 4: Init procedure performed by process $p_i$**

```
AERIAL SCANNING(p_i, v)
 1  if(status_i ≠ ⊥) then COVER(p_i)
 2  else broadcast(coverREQ, v, i)
 3    wait for 2 rounds
 4    if ∃ (coverRES, q, status_q, agent_q.ID, IMAGES_q) ∈ B_i ::
 5              status_q == v
 6    then   if(agent_i.ID ≠ ⊥) then UpdateAgent(IMAGES_q, v);
 7      if (agent_q.ID ≠⊥) ⋀(agent_i.ID =⊥) ⋀(waitAgent_i ==⊥)
 8      then waitAgent_i ← agent_q.ID;
 9           broadcast(agentREQ, agent_q.ID, i);
10           wait for 2 rounds
11           if receive(agentRES, agent_q, i);
12           then agent_i ← agent_q;
13           waitAgent_i ←⊥;
14      MOVE_VICINITY(v);
15    else
16      if (i == min(B'_i))
17        then status_i ← v;
18             if (agent_i ≠⊥)
19             then agentdone_i ← false
20             COVER(p_i)
```

**Figure 5: Aerial Scanning procedure performed by $p_i$ in vicinity $v$**

If a node is a covering node it simply invokes the COVER procedure (line 1 of Figure 5). The COVER procedure (see Figure 6), makes the covering node that executes it to move at take pictures of the grid points of the vicinity it covers (lines 1-6 of Figure 6). Each time a new picture is taken, $p_i$ updates its agent with the new information (if any) and its global picture (lines 8, 9 of Figure 6).

The UpdateAgent procedure invoked in the COVER procedure is shown in Figure 7.

The agent's COLLECTION is updated with the new images of the current vicinity (line 2 of Figure 7). These new images replace the corresponding old images. To generate a consistent picture, for each vicinity $v$ the agent needs to store in $COLLECTION[v]$ all last $k + T''$ images taken by the covering node at $v$. But, it is possible that when the agent collect the images of a vicinity from the corresponding covering

```
COVER(p_i)
1  j ← 1;
2  Repeat
3    IMAGES_i(j mod (k + T'')) ←< take_picture(l), time >;
4    move_to_grid(l, v);
5    l ← next(l, v);
6    j + +;
7    if (agent_i.ID ≠⊥) then
8      UpdateAgent(IMAGES_i, v);
9      UpdateGlobalImage(COLLECTION_{agent_i.ID})
10     agentmove_i ← true
11 Forever
```

**Figure 6: Cover procedure at node $p_i$ covering vicinity $v$.**

node, the covering node has not collected enough history of images of the vicinity. In this case, the agent moves on with any existing info and gets the rest in the next rounds.

```
UPDATEAGENT(IMAGES, v)
1   if ((agentdone_i == false) ⋀(agent_i.ID ≠⊥)
2                ⋀(IMAGES ≠ [⊥, ... ⊥]))
3   then COLLECTION_{agent_i.ID}[v] ← IMAGES
4        agentdone_i ← true
```

**Figure 7: Local computation performed by a covering node $p_i$.**

When a vagabond node $p_i$ enters a vicinity $v$, it executes the pseudocode in Figure 5. First, it investigates whether vicinity $v$ is covered. It broadcasts a cover request message *(coverREQ,v,i)* containing the identifier of the vicinity $v$ and the identifier $i$ of the requesting node $p_i$. Then $p_i$ waits for 2 rounds to collect cover response messages (lines 2-3 of Figure 5). According to our assumption on the broadcast communication primitive (see Section 3), within two rounds from the broadcast event, $p_i$ will receive a response of each node which was in vicinity $v$. We assume all such response messages (broadcast in line 3 of Figure 8) to be atomically received by $p_i$ at the beginning of the second round. $B_i$ is the buffer containing all such cover response messages, (line 4 of Figure 5). Each message buffered in $B_i$ has the following format:

$(coverRES, q, status_q, agent_q.ID, IMAGES_q)$ where $q$ is the identifier of the sender node, $status_q$ states if the sender covers some vicinity and which vicinity it covers; $agent_q.ID$ is the identifier of the agent incarnated by node $p_q$ if any; and $IMAGES_q$ are the images collected by node $p_q$ about the vicinity it covers.

If there exists a message in $B_i$ whose $status_q$ is equal to $v$, then there exists a node $p_q$ which is currently covering vicinity $v$ (line 4 of Figure 5). In this case, node $p_i$ continues to be vagabond and has to move to the next vicinity by calling MOVE_VICINITY (line 14 of Figure 5). Before moving, $p_i$ checks (in line 7 of Figure 5) whether both $p_q$ incarnates an agent (i.e., $agent_q.ID \neq \perp$) and it does not already incarnate another (i.e., agent $agent_i.ID = \perp$). If so, $p_i$ requests to incarnate $p_q$'s agent (line 9 of Figure 5) and waits for 2 rounds for a response. If it has been selected, it takes the agent (lines 11-12 of Figure 5). If $p_i$ already incarnates an agent, it simply updates its agent with the images taken by the covering node (line 6 of Figure 5).

Otherwise, vicinity $v$ is uncovered and a node must be elected to cover $v$. Note that several vagabond nodes can concurrently enter the same vicinity and concurrently ask to cover it. Concurrent requests (to check the status of vicinity $v$) generated by multiple vagabond nodes are lexicographically ordered according to the identifiers of the nodes that broadcast the requests (line 16 of Figure 5). Each node that receives these requests (broadcast concurrently with its own request), it stores them in a buffer. Let $B'_i$ be this buffer for node $p_i$. Once the information stored in either $B_i$ or $B'_i$ is used (in lines 4 and 16 of Figure 5, respectively) then these buffers get empty.

```
1  Upon B''_q ≠ ∅
2  if (status_q == v)
3  then broadcast(coverRES, q, v, agent_q.ID, IMAGES_q)
```

**Figure 8: Receive thread at process $p_q$ to manage cover requests**

In Figure 8, we describe the response broadcast by a covering node that covers vicinity $v$ which is triggered by receiving at least one message of the form $(coverREQ, v, i)$ (for any $i$). Note that it suffices that only one broadcast happens even if multiple such messages are received concurrently. Such multiple requests are stored locally by node $p_q$ in a buffer $B''_q$ which gets empty immediately after it triggers the procedure described above (see Figure 8).

Figure 9 shows the thread run by a covering node $p_q$ when it receives requests (from vagabond nodes) for the agent it incarnates. Recall that all the requests broadcast at some round $j$ are received in burst at the beginning of round $j + 1$. For sake of simplicity, we denote $AMB$ the burst of agent request messages which are of the form: $(agentREQ, agentID, j)$, (for any $j$) received at the beginning of each round (line 1 of Figure 9). Node $p_q$ checks if the agent requested is the one it incarnates (line 2 of Figure 9). Then, it chooses among the concurrent requests, the node that will take care of its agent (line 4 of Figure 9). This latter is the node with the minimum identifier among the nodes that (concurrently) requested the agent.

```
1  Upon receive (AMB)
2  if (agent_q.ID == agentID)
3  then UpdageAgent(IMAGES_q, v);
4     broadcast(agentRES, agent_q, min(AMB))
5     agent_q ←⊥
```

**Figure 9: Receive thread run at covering node $p_q$ to manage agent request messages from vagabond nodes.**

When the vagabond node $p_i$ is ready to move to the next vicinity of $v$ it calls the procedure MOVE_VICINITY($v$) (which appears in Figure 10). If node $p_i$ has an agent, then before it physically moves (by calling $move\_to\_vicinity(v)$ in line 8 of Figure 10) it broadcasts its agent information so that all other (covering or not) nodes that receive it can update their global image accordingly (Figure 11). It also updates its own global image (line 3 of Figure 10). Otherwise, if $p_i$ has no agent, it attempts to get refreshed images of vicinity $v$ from $v$'s covering node (lines 4-7 of Figure 10).

```
MOVE_VICINITY(v)
1  if (agent_i.ID ≠⊥)
2  then  broadcast(COLLECTION_{agent_i.ID})
3        UpdateGlobalImage(COLLECTION_{agent_i.ID});
4  else  broadcast(InfoREQ, v)
5        wait for 2 rounds
6        if receive(InfoRES, INFO, v)
7        then UpdateGlobalImage(INFO)
8  move_to_vicinity(v)
9  if (agent_i ≠⊥)
10 then agentdone_i ← false
11 AerialScanning(p_i, nextvicinity(v))
```

**Figure 10: Procedure to transfer vagabond node $p_i$ to next vicinity**

The response from the covering node contains either the covering node agent's COLLECTION (line 4 of Figure 12) if it has an agent, or the covering node's global picture (line 5 of Figure 12). Note that if the vicinity $v$ has no covering node, then $p_i$ simply does not perform any update. Once the above is complete, $p_i$ physically moves to the next vicinity of $v$ (line 8 of Figure 10) and if it has an agent it sets its agentdone$_i$ flag to false (lines 9, 10 of Figure 10). The latter is performed so that its agent gets updated with the new images of the newly visited vicinity (this happens in line 6 of the procedure Aerial Scanning of Figure 5). After all these are completed, then the node calls the Aerial Scanning procedure (Figure 5) to perform all necessary checks in the new vicinity where it is now located.

```
1  Upon receive (COLLECTION_{a_j})
2  UpdateGlobalImage(COLLECTION_{a_j})
```

**Figure 11: Global Image update thread at node $p_q$**

```
1  Upon receive (InfoREQ, v)
2  if (status_q == v)  then
3  if  (agent_q ≠ ⊥)
4       then broadcast(InfoRES, COLLECTION_{agent_q.ID}, v)
5       else broadcast(InfoRES, G_q, v)
```

**Figure 12: Thread performed at node $p_q$ to manage the requests of Global Picture information**

When the agent in a covering node $p_i$ has been updated and its information has been used to update the global picture at $p_i$, the agent can move to the next vicinity. This is triggered by setting the variable $agentmove_i$ to true (line 10 of Figure 6) which causes $p_i$ to execute (in parallel with its duty to move and take pictures of its covering vicinity) the procedure AGENT_MOVE (see Figures 13 and 14).

To perform AGENT_MOVE, $p_i$ keeps asking (by broadcasting a request) the nodes in the next vicinity whether its covering node exists and also it does not hold an agent. Then this covering node could accept its agent (lines 1-4 of Figure 14). This continues until either a vagabond node of its vicinity or the covering node of the next vicinity accepts the agent. This request may happen multiple times because first, the next vicinity may be uncovered, and second, it may be

```
1  Upon agentmove_i == true
2  AGENT_MOVE(p_i, v)
```

**Figure 13: Thread performed at $p_i$ covering vicinity $v$ to call AGENT_MOVE procedure.**

covered but its covering node hosts an agent and hence it is not able to accept another one. Either of these cases will be eventually resolved as we prove in Section 4.2. When a new hosting node is found, the agent is transferred (lines 5-6 of Figure 14). Variable agent$_i$ gets updated (line 7 of Figure 14) because $p_i$ no longer has an agent. Also (line 8 of Figure 14) the variable agentmove$_i$ is set to false.

```
AGENT_MOVE(p_i, v)
1  repeat
2    broadcast(MoveAgent, agent_i.ID, v)
3    wait for 2 rounds
4  until ((agent_i.ID ==⊥) ⋁(receive(MoveAgentRES, a_i, q))
5  if receive(MoveAgentRES, agent_i.ID, q)
6  then broadcast(agent_i.ID, COLLECTION_{agent_i.ID}, q)
7  agent_i ← (⊥, [⊥, . . . , ⊥], . . . , [⊥, . . . , ⊥])
8  agentmove_i ← false
```

**Figure 14: Procedure performed by covering node $p_i$ to transfer its agent.**

When a node $p_q$ that is covering next($v$) receives the request to incarnate an agent currently in vicinity $v$, it checks if it already incarnates another agent or if it is already waiting for another agent (lines 1-3 of Figure 15). If not, then it can incarnate an agent. To do so, it sets its waitAgent variable to the ID of the agent it is waiting for and sends its acceptance to the node that currently hosts the given agent (line 5-6 of Figure 15). Then $p_q$ waits for 2 rounds for a response by the agent's current host. If in the meanwhile $p_q$ receives a message notifying it that it has been elected to incarnate the agent (line 8 of Figure 15), it updates its variables related to the agent management (lines 10-11 of Figure 15). Finally, variable waitAgent gets value $\perp$.

```
1  Upon receive (MoveAgent, a_j, v)
2  if ((status_q == nextvicinity(v)) ⋀(waitAgent_q ==⊥)
3            ⋀(agent_q ==⊥))
4  then
5    waitAgent ← a_j
6    broadcast(MoveAgentRES, a_j, q)
7    wait for 2 rounds
8    if receive(a_j, COLLECTION_{a_j}, q)
9    then
10       agent_q ← (a_j, COLLECTION_{a_j})
11       agentdone_q ← false
12   waitAgent_q ←⊥
```

**Figure 15: Pseudocode performed by node $p_q$ when it receives a request to move an agent.**

We note that during the AGENT_MOVE procedure it is possible that the node that has the agent fails. We do not need to take care of this case during AGENT_MOVE because if it happens we assume that then simply the agent has failed. This does not cause a problem since we have $f + 1$ agents and even if $f$ of them fail one will be still in the system.

## 4.2 Correctness Proof

In this section, we prove correctness of our aerial scanning algorithm.

LEMMA 4.1. *At all times, there is at most one covering node covering each vicinity.*

PROOF. Initially, the lemma holds because we assign exactly one node to cover each vicinity. This initial assignment can only change after a failure happens and, as a result, a vicinity remains uncovered. Assume that at some time $t$ an uncovered vicinity $v$ gets covered by some new node $p_i$ (that was a vagabond node). The request for this to happen is broadcast by $p_i$ at round $t-2$ (line 2 of Figure 5). Assume that a set of (vagabond) nodes $Q$ broadcast such requests at time $t-2$. All such request will be received by all (not failed) nodes in $Q$ at time $t-1$. Since $p_i$'s request is the smallest, no other node in $Q$ will become a covering node of the uncovered vicinity. If at least one node asked to cover this vicinity (by broadcasting a request with its identifier) before round $t-2$, then unless this node has failed, the vicinity would be covered by round $t-1$. But then, at round $t-1$, the covering node would have disallowed $p_i$ to become a covering node as well, by broadcasting a message that contains (coverRES, $v$) which would be received by $p_i$ by round $t$. If a node broadcasts a request to cover $v$ after round $t-2$ then if $p_i$ has not failed, $p_i$ would disallow it to become a covering node by broadcasting a message that contains (coverRES, $v$) at some time at or after $t$. This message would be received by the trying node during the 2 rounds it is waiting (line 3 of Figure 5), and hence, it will not become a covering node. □

LEMMA 4.2. *It would take at most $6+\sqrt{k}$ rounds for a vagabond node to move from one vicinity to the next (adjacent) vicinity in the order specified by cycle $C$.*

PROOF. To physically move from one vicinity to another it wold take at most $\sqrt{k}$ rounds. This is the cost of move_to_vicinity() which is called in MOVE_VICINITY (line 8 of Figure 10). Therefore, the cost of MOVE_VICINITY is $2+\sqrt{k}$ (2 rounds for waiting in line 5 of Figure 10). According to the algorithm this can be delayed by another 4 rounds in the worst case. This is because it would take 2 rounds for a vagabond node waiting to collect messages about the status of the vicinity (line 3 of Figure 5) and if the vagabond node can take an agent it would take an additional 2 rounds to do so (line 10 of Figure 5). □

LEMMA 4.3. *It would take at most $T = (6+\sqrt{k})(m-1)$ rounds for a vagabond node to complete visiting all vicinities exactly once.*

PROOF. From Lemma 4.2, to perform a complete visit of all $m$ vicinities, it would take $(6+\sqrt{k})(m-1)$ rounds since the path $C$ they move on is a simple cycle of the $m$ vicinities. □

LEMMA 4.4. *It would take at most $T' = (6+\sqrt{k})(m-1)+2$ rounds for an uncovered area to become covered.*

PROOF. If there is an uncovered area that means that there is at least one vagabond node in the system. This is because there can be at most $m-1$ covered areas having at most one covering node each (by Lemma 4.1) and there are at least $m+f$ nodes in the system out of which at most $f$ of them can fail. This vagabond node will find the uncovered area within the $T$ rounds it would take it in the worst case to visit all vicinities (Lemma 4.2). It will take another 2 rounds to become its covering node (line 3 of Figure 5). □

For the next lemma, we assume that $n \geq 2f+2$ and that the parameter describing the frequency of failures $F$ is larger than $2(2T+10+4m+\sqrt{k})$. This implies that at most one failure can happen during any period of $2(2T+10+4m+\sqrt{k})$ rounds.

LEMMA 4.5. *An agent would require at most $T'' = 2(2T+10+4m+\sqrt{k})m$ rounds to complete visiting all vicinities exactly once.*

PROOF. First, we will calculate the time it takes for an agent in a node in vicinity $v$ at time $t$ to move to a node in the next vicinity $v'$. Let $\tau$ be this time. Then, since there are $m$ vicinities, the time an agent requires to visit all of them is $T'' = \tau m$.

To calculate $\tau$, we consider the following cases:

If the agent is on a vagabond node in $v$ at time $t$ it will remain on this vagabond node until this node fails. In this case, the agent moves together with the vagabond node. From Lemma 4.2, it would take $\tau = 6+\sqrt{k}$ rounds for the agent to move from $v$ to $v'$ on its hosting vagabond node. Otherwise, the agent is on the covering node of $v$ at time $t$. Since covering nodes do not move outside the covered vicinity, to move the agent in a new vicinity, we need to migrate this last from the current hosting node to a new node which is either vagabond or covering the next vicinity.

Because each node in each given round can incarnate at most one agent, it is necessary, that there is always a node without an agent. Otherwise, the agents would not possibly move away from their hosting nodes because there would not be any available node without an agent to accept them. The existence of a node without an agent follows by our assumption that $n \geq 2f+2$. Then, there are at least $f+2$ nodes in the system even if $f$ nodes have failed. Because there are at most $f+1$ agents at any time, this implies that at all times there is at least one node in the system without an agent.

Let $q$ be such a node (that we call an *empty node*) at time $t$. We have two possible subcases: $q$ is a covering node, or $q$ is a vagabond node. Let us assume that in the following time period $\Delta := [t, t+(2T+10+4m+\sqrt{k})]$ no failures happen.

- Assume that the empty node $q$ is a covering node. After $T'$ rounds (i.e., by time $t+T'$), no vicinity remains uncovered in $\Delta$ because of Lemma 4.4 and the assumption that no more failures happen during $\Delta$. By the algorithm, since no vagabond node ever gives an agent

to a covering node, there will always be an empty covering node during $\Delta$.

Consider two consecutive vicinities (in the ordering $C$) such that the first has a non-empty covering node and the second has an empty covering node. The non-empty covering node will try to move its agent to the following vicinity which may be waiting for some previous agent (i.e., its waitAgent variable is not equal to $\perp$). This will be resolved in exactly 2 rounds (line 7 of Figure 15) after which the empty covering node sets its variable waitAgent to $\perp$ (line 12 of Figure 15). Note, that at this point, waitAgent can only be set to the identifier of the agent hosted by the covering node of the previous vicinity. This is because there is only one covering node per vicinity, and the covering node of the empty vicinity would never get an agent from a vagabond node. After that, it will take at most another 2 rounds (line 3 of Figure 14) for the non-empty node to receive notification (i.e., a message containing MoveAgentRES), from the empty node. This will immediately trigger the non-empty node to broadcast a message containing the agent. This will cause the covering node of the preceding vicinity to become empty. Note that it is possible that this vicinity becomes empty earlier (than these 4 rounds) if an empty vagabond node visits it first. We conclude that, it will take at most 4 rounds for a non-empty covering node of some vicinity to become empty if the following vicinity has an empty covering node.

In the worst case, at most $(m-1)$ vicinities with non-empty covering nodes will have to become empty, before $v$ becomes empty. Hence, it can take up to $4m$ rounds for the agent in $v$ to be broadcast by its hosting covering node of $v$.

We conclude that, $v$ will send its agent to the empty covering node of $v'$ within at most $T' + 4m = T + 2 + 4m$ (i.e., $T'$ rounds to ensure that all vicinities are covered, $4m$ rounds to ensure that the covering node of $v$ is empty). Within that time it is also possible that a vagabond node will get $v$'s agent from its covering node. In either case, $v$'s covering node will be empty within $T + 2 + 4m$ rounds.

Next, we calculate the time it takes for the agent to move to $v'$. If the agent was sent to the covering node of $v'$ (line 6 of Figure 14) then it will only take one additional round for $v'$ to receive the message containing the agent (line 8 of Figure 15). Otherwise, if $v$'s agent went to a vagabond node, it will take an additional $6 + \sqrt{k}$ rounds for this agent to move to $v'$ on this vagabond node (by Lemma 4.2). We conclude that in the worst case, the agent will be moved from $v$ to $v'$ within at most $\tau = (T + 2 + 4m) + (6 + \sqrt{k}) = (T + 8 + 4m + \sqrt{k})$ rounds.

- Otherwise, the empty node $q$ is a vagabond node. Since there is at least one non-empty covering node in the system (i.e., the covering node of $v$), then some empty vagabond node will get the agent from a non-empty covering node and cause it to be empty. This will happen within $T$ rounds it takes for the empty vagabond node to visit all vicinities (Lemma 4.3) and an additional 2 rounds to wait for the requested agent (line 10 of Figure 5). Therefore, within $T + 2$ rounds, some covering node will be empty and, from the previous case, it will take another $(T + 8 + 4m + \sqrt{k})$ rounds for $v$ to transfer its agent to $v'$ (either on a vagabond node, or from covering node of $v$ to covering node of $v'$). In total, $\tau = (2T + 10 + 4m + \sqrt{k})$.

From the above case analysis, if no failures happen during $\Delta$, we conclude that $\tau = \max((6+\sqrt{k}), (T+8+4m+\sqrt{k}), (2T+10+4m+\sqrt{k}) = 2T+10+4m+\sqrt{k}$. Therefore, if no failures happen during $\Delta = [t, t + (2T + 10 + 4m + \sqrt{k})]$, $v$ will be able to transfer its agent to $v'$ within at most $(2T+10+4m+\sqrt{k})$ rounds. If failures can happen during $\Delta$, then because $F > 2(2T + 10 + 4m + \sqrt{k})$, no failures will happen during $[t + (2T + 10 + 4m + \sqrt{k}), t + 2(2T + 10 + 4m + \sqrt{k})]$. The agent is guaranteed to move from $v$ to $v'$ within this time by the same arguments applied for time $t+(2T+10+4m+\sqrt{k})$ instead of time $t$. Hence, we conclude that in either case, the agent will be able to move from $v$ to $v'$ within at most $\tau = 2(2T + 10 + 4m + \sqrt{k})$ rounds. □

To prove correctness of our algorithm, we slightly decrease the frequency of failures by choosing $F > 2(T'+k+3T''+3)$.

THEOREM 4.6. *Properties 2.1, 2.2 and 2.3 hold.*

PROOF. Assume that no failures happen during a period of time of duration $T' + k + 3T'' + 3$. Within the first $T'$ rounds of this period all vicinities have a covering node by Lemma 4.4. Within the following $k + T''$ rounds all covering nodes will have collected images of their vicinities for the last $k+T''$ rounds. These images are stored in variable IMAGES$_i$ of the covering node $p_i$ as described in line 3 of Figure 6.

Within the following $T''$ rounds at least one agent has moved around visiting all covering nodes (by Lemma 4.5). During this travel, it will collect the last $k + T''$ images stored in each covering node at the time it visits it. Within another $T''$ rounds it will perform another pass of all vicinities and broadcast messages with the information it has collected so far (which contains the $k + T''$ last images encountered for each vicinity).

The agent gives the information it has to a covering node either in line 2 of Figure 10 if the agent is on a vagabond node, or in line 9 of Figure 6 if the agent if on the covering node. If the agent was on a vagabond node, then it takes one additional round for the covering node to receive the information and update its global picture (Figure 11). In either case, after at most one additional round each covering node will be able to construct a *correct* global image of the area of interest which satisfies properties 2.1 and 2.2 as we explain next. Property 2.1 holds because of the way we divided the area into vicinities. Property 2.2 holds because by collecting sequences of images of each vicinity taken during the last $k + T''$ rounds the nodes that get them during the $T''$ time it takes to distribute them, they can always find a subsequence of $k$ images (one such set for each vicinity) that are taken within the same subperiod of $k$ consecutive time units. We

conclude that within $T' + k + 3T'' + 1$ rounds all covering nodes have the correct global image.

Similarly, each vagabond node with an agent will also be able to get a correct global image from the information it can take from its own agent within $T' + k + 3T'' + 1$ rounds.

Finally, any vagabond node without an agent can get the correct global image that is stored in the covering node of the vicinity it is in. This will happen within another 2 rounds after all covering nodes have the correct global image (i.e., within $T' + k + 3T'' + 3$ rounds). This is performed in the procedure MOVE_VICINITY (line 5 of Figure 10). This information is guaranteed to get to the vagabond node within 2 additional rounds because no failures happens, all vicinities will be covered, and all covering nodes have already collected the appropriate information.

We conclude that if no failure happens during a period of time of duration $T' + k + 3T'' + 3$, then it would take up to $T_{total} = T' + k + 3T'' + 3$ rounds for all nodes to get a correct image. Because $F > 2T_{total}$, then if one failure happens during any $T_{total}$ rounds, it is guaranteed that no failure will happen in the following $T_{total}$ rounds. Then, during the second period of $T_{total}$ rounds, all nodes will get correct global images as explained above. We conclude that Property 2.3 holds for any $h > 2T_{total}$ because each node can get a correct global image at least once every $2T_{total}$ rounds. □

## 5. RELATED WORK

Several solutions to aerial scanning of a geographic area using UAVs have been provided in the literature. To the best of our knowledge all current solutions rely on UAVs to continuously take images of some geographic area. The global image is composed like a mosaic of smaller images and this is done by a mobile or static ground station [4], [8], [2], [7].

On the contrary, we provide a distributed solution where UAVs cooperate to share a frequently refreshed complete image of the geographic area of interest. We do not address aspects related to image distortion caused by the physical properties of cameras mounted of the UAV. These aspects are treated in several papers in the literature, e.g., [4], [3].

Our solution coordinates the UAVs so that in total they cover the whole geographic area of interest. Additionally, the UAVs spread the images they take so that each can create a global image that will be a mosaic of some of the images received. To do so, we developed a virtual agent which travels through the area collecting and spreading information to UAVs. The virtual agent can be seen as a token continually circulating between some subareas (that we called vicinities) in the mobile ad-hoc network. In [5] token circulation algorithms are presented for the mobile ad-hoc network. These latter request the token to pass through all the nodes in the system, while in our case, either the agent moves together with a node traveling between vicinities or in the worst case, it only goes through one node for each vicinity. Hence, our solution is efficient due to the reduced amount of messages that flow the network and that have to be processed by each UAV compared to token-passing solutions. Distributing information can be achieved by performing global broadcasting [6]. This simple solution would continuously flood the network with messages. In a UAV system, energy consumption is strategic due to their reduced physical resources. This makes expensive solutions not feasible. Our solution only uses local broadcast to achieve the necessary communication.

## 6. CONCLUSION AND FUTURE WORK

In this paper, we formally defined and solved the aerial scanning problem using UAVs (i.e., unmanned aerial vehicles). The model we considered is a mobile ad-hoc network. Unlike previous solutions, we did not assume the existence of a central station which collects images of subareas taken by UAVs. Instead, the global images constructed must be frequently available to all UAVs in the system. Our solution is fault tolerant and it is designed so that it exclusively uses local communication which is inexpensive. This is important when dealing with mobile UAVs which have limited capabilities. The aerial scanning problem is useful for numerous practical applications making efficient solutions valuable. We did propose an efficient solution to this problem. A natural question which arises is whether our solution is optimal considering different metrics such as communication cost, or frequency of updates of the global image. To our knowledge such important questions have not been answered considering the aerial scanning problem in the setting that we have proposed.

## 7. REFERENCES


[1] C.Koo, V. Bhandari, J. Katz, and N. H. Vaidya. Reliable broadcast in radio networks: the bounded collision case. In *PODC*, pages 258–264, 2006.

[2] E. Kuiper and S. Nadjm-Tehrani. Mobility models for uav group reconnaissance applications. In *ICWMC*, page 33, 2006.

[3] Y. Lin, Q. Yu, and G. Medioni. Map-enhanced uav image sequence registration. In *In proceedings of 2007 IEEE Workshop on Applications of Computer Vision (WACV'07)*, page 15, 2007.

[4] J. Majumdar, S. Vinay, and S. Selvi. Registration and mosaicing for images obtained from uav. In *International Conference on Signal Processing and Communications SPCOM'04*, pages 198– 203, 2004.

[5] N. Malpani, Y. Chen, N. Vaidya, and J. Welch. Distributed token circulation in mobile ad hoc networks. In *IEEE Transactions on Mobile Computing*, volume 4, pages 154– 165, 2005.

[6] M. Mohsin, D. Cavin, Y. Sasson, R. Prakash, and A. Schiper. Reliable broadcast in wireless mobile ad hoc networks. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 233.1, Washington, DC, USA, 2006. IEEE Computer Society.

[7] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. Hedrick. An overview of emerging results in cooperative uav control. In *43rd IEEE Conference on Decision and Control*, pages 602–607, 2004.

[8] S. Spry, A. Girard, and J. Hedrick. Convoy protection using multiple unmanned aerial vehicles: organization and coordination. In *Proceedings of the 2005 American Control Conference*, volume 5, pages 3524 – 3529, 2005.