Distributed deployment of asynchronous guards in art galleries

Anurag Ganguli Jorge Cortés Francesco Bullo

***Abstract*— This paper presents deployment algorithms for multiple mobile robots with line-of-sight sensing and communication capabilities in a nonconvex polygon. The objective of the proposed algorithms is to achieve full visibility of the environment. We solve the problem by constructing a novel data structure called the vertex-induced tree and designing schemes to explore the nodes of the tree by means of distributed algorithms. The agents are assumed to have access to a memory and their operation is partially asynchronous.**

I. Introduction

Consider a group of point *guards* equipped with sensors with omnidirectional vision. The classical Art Gallery Problem is to find the smallest number of such guards necessary to guard a simple polygonal environment. In other words, the problem is to find the minimum number of guards and locate them in a nonconvex polygonal environment so that each point of the environment is visible to at least one guard; see Figure 1. This problem has inspired a whole class of geometric problems on art galleries; see for example the beautiful survey [1]. The well known Art Gallery Theorem [2] says that $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and occasionally necessary to guard a polygon with n vertices. Fisk's constructive proof of this theorem [3] provides an elegant way of finding the locations of the guards. However, there is no discussion of the problem of finding the locations in real time with limited knowledge of the environment.

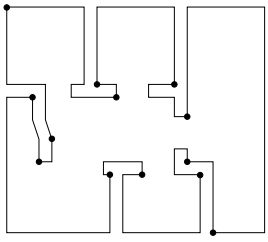

Fig. 1. A nonconvex polygon shaped like a typical floor plan: the solid circles represent the locations of *guards* with omnidirectional vision. Note that each portion of the environment is visible to at least one guard.

In this paper, we consider a group of robotic agents modeled as point masses, moving in a simple nonconvex polygonal environment. We assume that each member of the group is equipped with omnidirectional line-of-sight sensors. By a line-of-sight sensor, we mean any device or combination of devices that can be used to determine, in its line-of-sight, (i) the position or state of another agent, and (ii) the distance to the boundary of environment. By omnidirectional, we mean that the field-of-vision for the sensor is 2π radians. We also assume that each agent can communicate with any other agent visible to it and located with a certain distance. Given this model, the goal is to design a provably correct discrete-time algorithm which ensures that the agents converge to locations such that each point of the environment is visible to at least one agent. We shall henceforth refer to this particular problem as the *visibility-based deployment problem*.

This problem is related to many surveillance and pursuit-evasion problems in unknown environments. For example, imagine that one is interested in surveilling an unknown urban or an indoor environment. It is reasonable to model the environment as a nonconvex polygon. The problem of deploying robots to achieve full visibility is related to this objective. There has been a lot of research interest in the problem of deployment of robots in indoor environment for the purpose of surveillance. Some related works include [4] where an incremental algorithm for deployment is proposed and [5] in which the relevance of random walk on graphs is discussed. In addition, this problem is related to the pursuit-evasion problems in robotics; e.g., see [6]. This work is also noteworthy because of the use of an appropriate data structure to navigate through unknown polygonal environments.

The contribution of this paper is three-fold. First, given a simple nonconvex polygonal environment and one of its vertices, we describe a way to partition the environment into star-shaped polygons. This procedure leads us to define a new graph, associated to the nonconvex polygon and to the given vertex, called the vertex-induced tree. Every node in this graph corresponds to a star-shaped polygon. The edges between nodes are drawn whenever two star-shaped polygons have a common side. Second, we demonstrate how to plan paths between neighboring nodes of the vertex-induced tree based on a limited sensing and communication model. Third, we present two asynchronous and distributed algorithms for multiple agents to explore the nodes of the vertex-induced tree, and thereby solve the visibility-based deployment problem. In the design of these algorithms, we assume that the agents have some limited memory and are initially placed at the same location.

Notation: We begin by introducing some basic notation. We let $\mathbb{R}$ and $\mathbb{R}_+$ represent the set of real numbers and the set of nonnegative real numbers respectively. For $p \in \mathbb{R}^2$, let

Anurag Ganguli is with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, and with the Department of Mechanical and Environmental Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106, USA, aganguli@uiuc.edu

Jorge Cortés is with the Department of Applied Mathematics and Statistics, University of California at Santa Cruz, Santa Cruz, CA 95064, USA, jcortes@ucsc.edu

Francesco Bullo is with the Department of Mechanical and Environmental Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106, USA, bullo@engineering.ucsb.edu

$\overline{B}_r(p)$ denote the *closed ball* centered at p of radius $r \in \mathbb{R}_+$. Also, we let $\mathbb{N}$ refer to the set of natural numbers. Given two points $x, y \in \mathbb{R}^2$, we let $[x, y]$ represent the *closed segment* between x and y. Similarly, (x, y) represents the *open segment* between x and y, $[x, y)$ represents the set $(x, y) \cup \{x\}$ and $(x, y]$ represents the set $(x, y) \cup \{y\}$. Given a finite set X, let $|X|$ represent the cardinality of the set.

Now let us turn our attention to the polygonal environment. Let Q be a simple polygon, possibly nonconvex. We say that a polygon is simple if the polygon vertices are the only points in the plane common to two polygon edges and every polygon vertex belongs to at most two polygon edges. Such a polygon has a well defined interior and exterior. *Note that a simple polygon can contain holes*. Let $\mathcal{Q}$ refer to the set of all simple polygons. Let $\mathrm{Ve}(Q) = (v_1, \ldots, v_n)$ be the list of vertices of Q ordered counterclockwise. Throughout this paper, we shall reserve the use of the notation, n, to denote the number of vertices of a polygonal environment.

A point $q \in Q$ is *visible from* $p \in Q$ if $[p, q] \subset Q$. The *visibility polygon* $S(p) \subset Q$ from a point $p \in Q$ is the set of points in Q visible from p. It is convenient to think of $p \mapsto S(p)$ as a map from Q to the set of polygons contained in Q. It must be noted that the visibility polygon is not necessarily a simple polygon. Also, we shall use P to refer to tuples of elements in $\mathbb{R}^2$ of the form $(p_1, \ldots, p_N)$. With a slight abuse of notation, we shall use P interchangeably with a point set of the form $\{p_1, \ldots, p_N\}$.

II. Asynchronous network of visually-guided agents

We begin by introducing the notions of a *visually-guided agent* and of a *network of visually-guided agents*. By a visually-guided agent, we mean any agent with the following capabilities:

sensing: each agents is equipped with omnidirectional sensors with the ability to sense the relative position of any other agent within its line-of-sight and the also relative position of any point on the boundary of the environment that is visible from its location.

communication: each agent can communicate with any other agent that is within its line-of-sight and lying inside a closed disk of radius $r \in \mathbb{R}_+$. Note that (a) r is bounded from above by, say $R \in \mathbb{R}_+$, the maximum signal strength that an agent can send and, (b) at any time an agent can choose any $r \leq R$ by changing the signal strength. Hence if $p \in Q$ represents the position of any agent, then its communication region with a given radius $r \in \mathbb{R}_+$ is given by $\mathcal{C}(p) = S(p) \cap \overline{B}_r(p)$.

processing: ability to perform computations on the variables stored in its memory.

maneuvering: ability to move between two points.

A collection of such agents forms a network. Let N be the number of agents in the network. Each agent has a unique identifier (UID), $i \in \{1, \ldots, N\}$, associated with it. We also associate the following quantities to every agent i:

(i) an increasing sequence of time instants $T^i = \{t_l\}_{l \in \mathbb{N}} \subset \mathbb{R}_+$, called the wake-up schedule; each instant is called a wake-up instant;

(ii) the position, $p_i(t) \in Q$ with $t \in T^i$, of the agent in the environment.

In addition, each agent may require certain memory depending upon the task that it is performing or on the algorithm it is executing. For each agent i, let $\mathcal{M}_i(t)$ represent the contents of its memory at any time t.

By communicating with any other agent, we mean that any agent, i, can perform the following:

(i) $\mathrm{BROADCAST}_i(i, \mathcal{M}_i)$: Broadcast to all agents in its communication region its UID, i, and the states in its memory, $\mathcal{M}_i$.

(ii) $\mathrm{RECEIVE}_i(j, \mathcal{M}_j)$: Receive a broadcast from another agent, j.

We assume here that for any agent, i, there is a bounded time delay between a $\mathrm{BROADCAST}_i(i, \mathcal{M}_i)$ *message sent by the agent and the corresponding* $\mathrm{RECEIVE}_j(i, \mathcal{M}_i)$ *message received by another agent, j, in its communication region. Let $\delta > 0$ denote this upper bound.*

Now, for the sake of convenience we define various actions that any agent is capable of performing. Note that this may be formalized by means of a finite state machine.

SPEAK: This represents the state of an agent, i, when it sends a $\mathrm{BROADCAST}_i(i, \mathcal{M}_i)$ message;

LISTEN: This represents the state of an agent, i, when it is capable of accepting a $\mathrm{RECEIVE}_i(j, \mathcal{M}_j)$ message for some $j \in \{1, \ldots, N\}, j \neq i$;

PROCESS: This represents the state of an agent performing computations on the state in its memory;

MOVE: This represents the state of the agent when it is moving.

Till now, in this section, we have described a general asynchronous network of visually-guided agents. In the following subsection, we describe a particular communication and state transition model that we use in this paper.

A. Asynchronous schedule

Any agent i assumes the following states between any two wake-up instants, T^i_l and T^i_{l+1} :

(i) SPEAK at times $t = T^i_l + k\delta$, where $k \in \mathbb{N} \cup \{0\}$, $T^i_l + k\delta < T^i_l + \lambda^i_l + \rho^i_l$, $\lambda^i_l \geq \delta$ and $\rho^i_l \geq 0$;

(ii) LISTEN during the time interval $[T^i_l, T^i_l + \lambda^i_l + \rho^i_l)$;

(iii) PROCESS during the interval $[T^i_l + \lambda^i_l, T^i_l + \lambda^i_l + \rho^i_l)$;

(iv) MOVE during the time interval $[T^i_l + \lambda^i_l + \rho^i_l, T^i_{l+1})$. If the agent decides not to move then $T^i_{l+1} = T^i_l + \lambda^i_l + \rho^i_l$.

Remarks 2.1: (i) Note that the sequence T^i is not prespecified. Given any wake-up instant T^i_l, the next wake-up instant T^i_{l+1} is decided based upon the time the agent spends in each of the states in between the two wake-up instants.

(ii) An agent is capable of receiving broadcasts always except when it is moving.

See Figure 2 for a schematic illustration of the above schedule.

Agent i, in the MOVE state, is capable of moving at any time $t \in [T^i_l + \lambda^i_l + \rho^i_l, T^i_{l+1})$ according to the following discrete-time control system:

$$p_i(t + \Delta t) = p_i(t) + u_i, \tag{1}$$

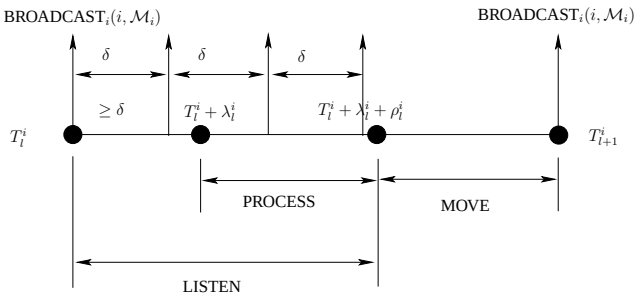

Fig. 2. Sequence of actions performed by an agent i in between two wake-up instants. Note that a BROADCAST$(i, \mathcal{M}_i)$ is an instantaneous event taking place where there is a vertical pulse, where as the PROCESS, LISTEN and MOVE actions take place over an interval. The MOVE interval might be empty if the agent does not move.

where the control is a function of the communication, sensing, memory and the action that the agent is performing at time instant t. Again, note that T_{l+1}^i is not predecided but is the time when the agent stops performing the MOVE action. Also, note that this model of visually-guided agents is similar in spirit to the *partially asynchronous model* described in [7].

III. The visibility-based deployment problem

We have introduced the Art Gallery Problem in Section I. The formulation of this classic problem assumes a priori knowledge of the polygonal environment. What we are interested in here is an *online* and *distributed* version of this problem.

Definition 3.1: Given a simple polygon Q, let $p_1(T_0^1), \ldots, p_N(T_0^N) \in Q$ represent the initial positions of an asynchronous network of N visually-guided agents as described in Section II. Let $\mathcal{A} : Q^n \to Q^n$ represent an algorithm which transforms a set of n points in Q to another set of n points in Q. We say that $\mathcal{A}$ solves the visibility-based deployment problem if the algorithm converges to a set $W \subset (2^Q)^N$ with the property that $\cup_i S(p_i) = Q$ for all $i \in \{1, \ldots, N\}$ and all $(p_1, \ldots, p_N) \in W$.

IV. The vertex-induced tree

Let us start by describing a procedure to partition* a polygonal environment without holes, Q, into star-shaped polygonal regions. To begin constructing this partition, we require a starting vertex which we call $s \in \mathrm{Ve}(Q)$. First, we introduce some notions.

Definition 4.1:
(i) Given a simple polygon X, any diagonal $[v', v'']$ of X where $v', v'' \in \mathrm{Ve}(X)$ partitions X into two polygons. If there exists a point $p \in X$ such that $p \notin [v', v'']$, then let $X_p(v', v'')$ represent the polygon containing p.

(ii) Given a simple polygon X and a vertex $v \in \mathrm{Ve}(X)$, let $(v_{u_1}, \ldots, v_{u_k})$ represent the list of vertices of X that are visible from v where $u_i < u_{i+1}$ for all $i \in \{1, \ldots, k-1\}$. Let $\mathcal{V}(v, X)$ be the polygon defined by this list. Clearly, this is a star-shaped polygon such that all points $q \in \mathcal{V}(v, X)$ are visible from v.

(iii) A *gap* of the polygon $\mathcal{V}(v, X)$ is a segment $[v_{u_j}, v_{u_{j+1}}]$ where $u_{j+1} > u_j + 1$. Thus each gap is a diagonal of X and divides it into two polygonal regions, one containing v, say $X_v(v_{u_j}, v_{u_{j+1}})$, and the other, given by $\overline{X \setminus X_v(v_{u_j}, v_{u_{j+1}})}$, not containing v.

In what follows, let G be a list whose elements are segments of the form $[x, y] \subset X$ with $x, y \in \mathrm{Ve}(X)$. Let K and $\mathcal{N}$ be lists whose elements are vertices, $v \in \mathrm{Ve}(X)$. Let $\mathcal{P}$ be a list whose elements are simple polygons. The elements G_i, K_i, $\mathcal{P}_i$ and $\mathcal{N}_i$ refer to the ith element in the lists G, K, $\mathcal{P}$ and $\mathcal{N}$ respectively. Also, n_G and n_K refer to the number of elements in the lists G and K respectively.

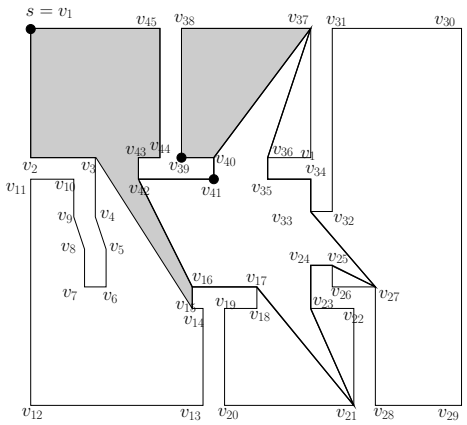

Fig. 3. Notations in Definition 4.1. Let the nonconvex polygonal environment represent X. $\mathcal{V}(s, X)$ is the shaded polygon in the figure represented by the vertex list $\{v_1, v_2, v_3, v_{15}, v_{16}, v_{42}, v_{43}, v_{44}, v_{45}\}$. The gaps associated with $\mathcal{V}(s, x)$ are $[v_3, v_{15}]$ and $[v_{16}, v_{42}]$ because $15 > 3 + 1$ and $42 > 16 + 1$. The gap $[v_{16}, v_{42}]$ partitions X into two polygons $X_s(v_{16}, v_{42})$, represented by the vertex list $\{v_1, \ldots, v_{16}, v_{42}, \ldots, v_{45}\}$, and $\overline{X \setminus X_s(v_{16}, v_{42})}$, represented by the vertex list $\{v_{16}, \ldots, v_{42}\}$. Note also that $v_{41} = \arg\min\{\|v - [v_{16}, v_{42}]\| \mid [v_{16}, v_{42}]$ is visible from $v, v \in \overline{X \setminus X_s(v_{16}, v_{42})}\}$.

Figure 3 illustrates the notations introduced above. The procedure then is as follows:

(i) Initialization:
  a) $G := \emptyset$; $K := \emptyset$; $\mathcal{P} := \emptyset$; $\mathcal{N} := \emptyset$.
  b) $X := Q$.
  c) Compute $\mathcal{V}(s, X)$.
  d) Insert $\mathcal{V}(s, X)$ into $\mathcal{P}$.
  e) Insert s into $\mathcal{N}$.
  f) Compute all gaps of $\mathcal{V}(s, X)$ and insert them into G. Let the number of such gaps be n_s. Then insert n_s copies of s into the list K. Hence, $n_G = n_K = n_s$.

(ii) Star-shaped partitioning algorithm:
While $n_G > 0$, execute the following steps:
  a) For G_1, K_1, compute $s' = \arg\min\{\|v - \frac{G_1\|}{X \setminus X_{K_1}(v', v'')}\} \mid G_1$ is visible from $v, v \in$, where $[v', v''] = G_1$.
  b) $X = \overline{X \setminus X_{K_1}(v', v'')}$.
  c) Compute $\mathcal{V}(s', X)$.

*Recall that a partition of any set X is a collection of closed subsets X_i of X such that $X = \cup_i X_i$ and $\mathring{X}_i \cap \mathring{X}_j = \emptyset$ for all $i \neq j$, where $\mathring{X}_i$ is the interior of the set X_i.

d) Insert $\mathcal{V}(s', X)$ at the end of $\mathcal{P}$.
e) Insert s' at the end of $\mathcal{N}$.
f) Remove G_1, K_1 from lists G and K respectively.
g) Compute all gaps of $\mathcal{V}(s', X)$ and insert them at the end of the list G. Let the number of gaps of $\mathcal{V}(s', X)$ be $n_{s'}$. Then insert $n_{s'}$ copies of s' at the end of the list K.

Remarks 4.2: (i) The computation in step (a) of the algorithm is always well posed. To see this note that any element G_i of G is a diagonal of Q. Hence $\overline{X \setminus X_{K_i}}(v_i', v_i'')$ with $[v_i', v_i''] = G_i$ is always a simple polygon with at least three vertices and with G_i representing an edge of the polygon. Since any simple polygon can be triangulated with diagonals, there always exists a triangle with G_i as one side. It is trivial to check that the entire segment G_i is visible from the third vertex of this triangle.

(ii) We assume here that there is a unique vertex $v \in \mathrm{Ve}(\overline{X \setminus X_{K_i}}(v', v''))$ which is at minimum distance from G_i and from which the entire segment G_i is visible. This assumption is not at all restrictive since if such a vertex is not unique, then we can break the deadlock by choosing the vertex that is first in the counterclockwise arrangement of vertices of $\overline{X \setminus X_{K_i}}(v', v'')$, the first being the end point v' of G_i such that the other end point, v'', is on the clockwise side of v'.

(iii) At any stage of the algorithm $(\overline{X \setminus X_{K_i}}(v', v'')) = (\overline{Q \setminus Q_{K_i}}(v', v''))$. Hence $\mathcal{V}(s', X)$ at step (c) of the algorithm can be computed with the knowledge of G_i and $S(s')$.

As an outcome of the algorithm, $\mathcal{P}$ is the list of star-shaped polygons which partition Q. In addition, all points of $\mathcal{P}_i$ are visible from the vertex $\mathcal{N}_i$. With some abuse of notation, henceforth, given Q and a vertex $s \in Q$, we shall refer to this partition as $\mathcal{P}_Q(s)$ and to the node list as $\mathcal{N}_Q(s)$. Finally, we refer to $\mathcal{P}_Q(s)$ as the *vertex-induced partition*. The following lemma summarizes the important properties of the vertex-induced partition.

Lemma 4.3: Given a simple polygon Q without holes and any vertex $s \in \mathrm{Ve}(Q)$, the following are true:

(i) $\mathcal{P}_Q(s)_i$ is a star-shaped polygon for all i; and
(ii) the list $\mathcal{N}_Q(s)$ belongs to the kernel† of Q, or in other words, for any $p_i \in \mathcal{N}_Q(s)$, we have that $\mathcal{P}_Q(s)_i \subset S(p_i)$.

We now define a graph using this partition. We assume that the reader is familiar with some standard notions of graph theory.

Definition 4.4: Given a polygon $Q \in \mathcal{Q}$ and a vertex $s \in \mathrm{Ve}(Q)$, the *vertex-induced tree* $\mathcal{G}_Q(s)$, is the graph such that the vertex list is $\mathcal{N}_Q(s)$ and an edge exists between any two vertices $\mathcal{N}_i, \mathcal{N}_j \in \mathcal{N}_Q(s)$ if and only if there exists a segment $[x, y] = \mathcal{P}_i \cap \mathcal{P}_j$ with x, y distinct.

Note that by virtue of the construction of the vertex-induced tree, any segment $[x, y] = \mathcal{P}_i \cap \mathcal{P}_j$ is such that $x, y \in \mathrm{Ve}(Q)$, or in other words, $[x, y]$ is a diagonal of Q. Note also that $\mathcal{N}_1 = s$. We refer to s as the root of $\mathcal{G}_Q(s)$.

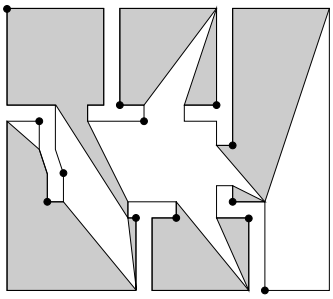

Fig. 4. The vertex-induced tree of a set of points in a nonconvex polygonal environment. The solid circles represent the vertices of the tree and the dashed lines represent the edges. The root of the tree is denoted by the vertex s.

We now state some important properties of the vertex-induced tree.

Lemma 4.5: Given a simple polygon Q without holes and any vertex $s \in \mathrm{Ve}(Q)$, the following statements are true:

(i) the graph $\mathcal{G}_Q(s)$ is a rooted tree;
(ii) no two nodes sharing an edge are visible to each other;
(iii) the maximum number of nodes in the vertex-induced rooted tree is $\lfloor \frac{n}{3} \rfloor$ where $n = |\mathrm{Ve}(Q)|$.

V. Algorithms

In this section we present motion planning algorithms to navigate between two points representing neighboring nodes in the vertex-induced tree. We then describe algorithms for multiple visually-guided agents to explore the nodes of the vertex-induced tree.

A. Navigation algorithms

Here we design algorithms to plan paths between neighboring nodes of the vertex-induced tree. Let us first state a lemma which characterizes the shortest path between any two neighboring nodes.

Lemma 5.1: Given a simple polygon Q without holes and any vertex $s \in \mathrm{Ve}(q)$, let $\mathcal{G}_Q(s)$ represent the vertex-induced tree and $\mathcal{N}_Q(s)$ the corresponding node set. Let $\mathcal{N}_Q(s)_i, \mathcal{N}_Q(s)_j$ represent two neighboring nodes and $\mathcal{P}_Q(s)_i \cap \mathcal{P}_Q(s)_j = [v', v'']$ where $v', v'' \in \mathrm{Ve}(Q)$ and $v' \neq v''$. Then the shortest path between $\mathcal{N}_q(s)_i$ and $\mathcal{N}_q(s)_j$ is given by the shorter of the two paths, $[\mathcal{N}_q(s)_i, v'] \cup [v', \mathcal{N}_q(s)_j]$ and $[\mathcal{N}_q(s)_i, v''] \cup [v'', \mathcal{N}_q(s)_j]$.

Any node of the vertex-induced tree has neighbors of possibly two types: parent or child. Let us first describe how to navigate from a node to its parent. Here are informal and formal descriptions of what we shall refer to as the *MOVE-TO-PARENT* routine:

> (i) Compute the shortest path between the parent and the node based on Lemma 5.1; (ii) go to the reflex vertex which is a part of the shortest path; (iii) from the reflex vertex go to the vertex representing the parent node.

†The kernel of a star-shaped polygon is the set of points from which the entire polygon is visible.

TABLE I

MOVE-TO-PARENT

Name: MOVE-TO-PARENT ($\{p_{\text{parent}}, p_{\text{last}}, v', v''\}$)
Goal: Go from node $\mathcal{N}_Q(s)_i$ to its parent, say $\mathcal{N}_Q(s)_j$
Requires: (i)$[v', v'']$ where $[v', v''] = \mathcal{N}_Q(s)_i \cap \mathcal{N}_Q(s)_j$, . (ii)$p_{\text{parent}} = \mathcal{N}_Q(s)_j$

1: $p_{\text{last}} := \mathcal{N}_Q(s)_i$
2: $p := \mathcal{N}_Q(s)_i$
3: Compute shortest path from p to $\mathcal{N}_Q(s)_j$, say $[p, v] \cup [v, \mathcal{N}_Q(s)_j$ where v is either v' or v''
4: **while** $p \neq \mathcal{N}_Q(s)_j$ **do**
5: **if** $p_{\text{last}} \neq v$ **then**
6: Compute shortest path from p to $\mathcal{N}_Q(s)_j$, say $[p, v] \cup [v, \mathcal{N}_Q(s)_j]$ where v is either v' or v''
7: $u = \frac{\min(s_{\max}, \|v-p\|)}{\|v-p\|}(v - p)$
8: **if** $u = 0$ **then**
9: $p_{\text{last}} = v$
10: **end if**
11: **else**
12: $u = \frac{\min(s_{\max}, \|\mathcal{N}_Q(s)_j - p\|)}{\|\mathcal{N}_Q(s)_j - p\|}(\mathcal{N}_Q(s)_j - p)$
13: **end if**
14: $p = p + u$
15: **end while**
16: **return**: $\{p_{\text{parent}}, p_{\text{last}}, v', v''\}$

Figure 5 shows paths between nodes and the respective parents as computed by the algorithm in Table I. Note that at the end of the MOVE-TO-PARENT routine the variable p_{last} is equal to the position of a vertex that belongs to the gap between the parent and the starting node. This information can be later used to decide which child of the parent should be visited next.

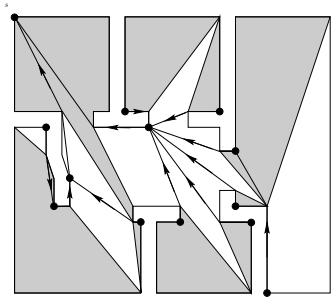

Fig. 5. The planned paths from nodes to their parent in the vertex-induced tree.

From Lemma 5.1 it can be seen that computing the shortest path between any two nodes requires the knowledge of the relative positions of one node with respect to the other. In the next section, we see that the algorithms that we design to explore the vertex-induced tree have the property that in order for any node to be visited by an agent, it must also have visited its parent. Hence to plan a path from a node to its parent, we can assume that the knowledge of the relative position of its parent is present. However, to plan a path from a node to any of its child, we cannot assume the knowledge of the relative position of the child. This is because the child might not have been visited by any other agent at all. However, the agents can detect the location of the gap between the node and the child. We now give informal and formal descriptions of what we shall refer to as the *MOVE-TO-CHILD* routine.

(i) Compute the mid-point of the gap between the node and the child; (ii) go to the mid-point; (iii) compute the nearest vertex from which the entire gap is visible and which is on the other side of the gap as the last node; (iv) go to that vertex.

TABLE II

MOVE-TO-CHILD

Name: MOVE-TO-CHILD ($\{p_{\text{parent}}, p_{\text{last}}, v', v''\}$)
Goal: Go from node $\mathcal{N}_Q(s)_i$ to its child, say $\mathcal{N}_Q(s)_j$
Requires: (i)$[v', v''] = \mathcal{N}_Q(s)_i \cap \mathcal{N}_Q(s)_j$

1: $p_{\text{last}} := \mathcal{N}_Q(s)_i$
2: $p := \mathcal{N}_Q(s)_i$
3: $p_{\text{temp}} = \frac{v'+v''}{2}$
4: **while** $p \neq p_{\text{temp}}$ AND $p_{\text{last}} \neq \frac{v'+v''}{2}$ **do**
5: **if** $p_{\text{temp}} \neq \frac{v'+v''}{2}$ **then**
6: $u = \frac{\min(s_{\max}, \|p_{\text{temp}} - p\|)}{\|p_{\text{temp}} - p\|}(p_{\text{temp}} - p)$
7: **if** $u = 0$ **then**
8: $p_{\text{temp}} = \arg\min\{\|v - [v', v'']\| \mid [v', v'']$ is visible from $v, \ , v \in X\}$, where $X = Q \setminus Q_{p_{\text{last}}}$
9: $p_{\text{last}} = \frac{v'+v''}{2}$
10: **end if**
11: **else**
12: $u = \frac{\min(s_{\max}, \|p_{\text{temp}} - p\|)}{\|p_{\text{temp}} - p\|}(p_{\text{temp}} - p)$
13: **end if**
14: $p = p + u$
15: **end while**
16: **return**: $\{p_{\text{parent}}, p_{\text{last}}, v', v''\}$

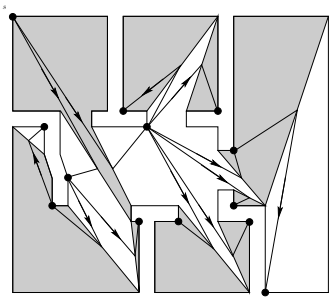

Fig. 6. The planned paths from nodes to their respective children in the vertex-induced tree.

Note that from the two algorithms in this section, the path from a node to its parent is shorter than the path from the parent to the node. Hence, we define the following notions.

Definition 5.2: Given a simple polygonal environment Q without holes, we define the following:

(i) the forward length of the graph $\mathcal{G}_Q(s)$, $\mathcal{L}_{\text{forward}}(\mathcal{G}_Q(s)) = \sum_{i=1}^{|\mathcal{N}_Q(s)|-1} \mathcal{L}_{\text{forward}}(\mathcal{G}_Q(s))_i$ with $\mathcal{L}_{\text{forward}}(\mathcal{G}_Q(s))_i =$ forward distance(e_i), where e_i is any edge of $\mathcal{G}_Q(s)$;
(ii) the backward length of the graph $\mathcal{G}_Q(s)$ $\mathcal{L}_{\text{backward}}(\mathcal{G}_Q(s)) = \sum_{i=1}^{|\mathcal{N}_Q(s)|-1} =$

$\sum_{i=1}^{|\mathcal{N}_Q(s)|-1} \mathcal{L}_{\text{backward}}(\mathcal{G}_Q(s))_i$ with
$\mathcal{L}_{\text{backward}}(\mathcal{G}_Q(s))_i = \text{backward distance}(e_i)$;

where e_i is an edge of $\mathcal{G}_Q(s)$ comprising of a node $\mathcal{N}_Q(s)_k$ and its parent $\mathcal{N}_Q(s)_l$. The length of the path from $\mathcal{N}_Q(s)_l$ to its child $\mathcal{N}_Q(s)_k$ is equal to forward distance(e_i) and the length of the path from $\mathcal{N}_Q(s)_k$ to its parent $\mathcal{N}_Q(s)_l$ is equal to backward distance(e_i).

B. Exploration algorithms

In this section, we present algorithms to solve a relaxed version of the visibility-based deployment problem. The additional assumptions we make here are that the agents have memory and that the initial positions of all the agents are the same. We also assume here that the environment has no holes. Note that by virtue of the construction in Section IV and the methods to navigate between one node of the vertex-induced tree to a neighboring node, we have converted the original problem into a graph exploration problem.

1) Exploring the vertex-induced tree: In this section, we design algorithms for multiple agents to cover the nodes of the vertex-induced tree under the assumption that all agents are initially located at the root of the tree. We present two algorithms to solve the problem. It must be noted that the algorithms we specify here may not be optimal in terms of performance measures such as required time. Our aim is mainly to give a solution that is guaranteed to solve the visibility-based deployment problem. Performance issues will be the subject of future research.

In Section II, we had talked about the communication region and the memory of an agent. Now, we specify what these are for the problem under consideration. For each agent i, we associate the following:

(i) The communication region specified by $\mathcal{C}(p_i) = S(p_i) \cap \overline{B}_{p_i}(r)$, where $r = \min\{R, \frac{1}{2}\min\{\|p_i - v\|,\ v \in S(p_i)\}\}$, if $p_i \in \text{Ve}(Q)$.

(ii) The list $\mathcal{M}_i(t)$ where each element is a point $p \in Q$. In the algorithms that we present later, the maximum number of required elements in the list is four. We let $\mathcal{M}_i(t)_k$ refer to the kth element of the list.

(iii) The list $\texttt{buffer-uid}_i$ whose elements are natural numbers.

(iv) The list $\texttt{buffer-memory}_i$ whose elements are lists of the type $\mathcal{M}$.

Let us first try to formally write the sequence of tasks performed by any agent in between two wake-up instants. The algorithm is described in Table III. Note that in the algorithm, we invoke two routines (i) depth-first search and (ii) randomized search which we describe later. For the present, we only inform the reader that they are decision making routines to execute the MOVE state.

The following lemma characterizes the set of agents whose messages are present in the buffer of any given agent.

Lemma 5.3: For any agent i at any time t, if $\texttt{buffer-uid}_i \neq \emptyset$, then $p_i(t) \in \mathcal{N}_Q(s)$ and there exists τ_j with $0 \leq \tau_j \leq \delta$ such that $p_j(t - \tau_j) = p_i(t)$ for all $j \in \texttt{buffer-uid}_i$.

TABLE III
ASYNCHRONOUS SCHEDULE

Goal:	Cover the nodes of the given vertex-induced tree, $\mathcal{G}_Q(s)$
Assumes:	$p_1(T_0^1) = p_1(T_0^2) = \ldots = p_N(T_0^N) = s \in \text{Ve}(Q)$

0: Assume k s.t. $\mathcal{N}_k = p_i$
0: $\texttt{buffer-uid}_i = \emptyset$
0: $\texttt{buffer-memory}_i = \emptyset$
0: $\mathcal{M}_i(T_0^i) = \{p_i(T_0^i), p_i(T_0^i)\}$
0: move-decision := `stay`

For each $i \in \{1, \ldots, n\}$, the following are executed at according to the schedule in Section II between any two wake up instants:

SPEAK
1: BROADCAST$_i(i, \mathcal{M}_i(t))$

LISTEN
1: RECEIVE$_i(j, \mathcal{M}_j(t-\tau))$, where $0 \leq \tau \leq \delta$
2: Append j to $\texttt{buffer-uid}_i$
3: Append $\mathcal{M}_j(t-\tau)$ to $\texttt{buffer-memory}_i$

PROCESS
1: run Depth-first search or Randomized search

MOVE
1: **switch** move-decision
2: **case** `stay`: Stay at $\mathcal{N}_k$
3: **case** `to-child`: $\texttt{buffer-uid}_i = \emptyset$; $\texttt{buffer-message}_i = \emptyset$; run MOVE-TO-CHILD($\mathcal{M}_i(t)$)
4: **case** `to-parent`: $\texttt{buffer-uid}_i = \emptyset$; $\texttt{buffer-message}_i = \emptyset$; run MOVE-TO-PARENT($\mathcal{M}_i(t)$)
5: **end switch**

We now present a standard depth-first algorithm to distribute the agents on the nodes of the vertex-induced tree. First, we give an informal description of the algorithm.

> Each agent performs the following tasks whenever the depth-first search routine is called: (i) Find the maximum UID among all agents which have communicated with it during the last δ units of time; (ii) If this UID is less than its own UID, then stay else move; (iii) If the decision is to move and there are no children of the present node, then move to parent; (iv) If the decision is to move and there is at least one child, then order the children in a suitable way. If the last node visited is the parent of the present node, then move to first child in the ordering. Otherwise, if the last node visited is a child that is not the last in the ordering,then move to the child that comes next in the ordering. Otherwise, if the last node visited is a child that is the last in the ordering, move to the parent node.

We formally describe the depth-first search routine in Table IV. Before presenting the correctness proof of this algorithm, let us present a randomized search routine. We first provide an informal description of the algorithm.

> Each agent performs the following tasks whenever the depth-first search routine is called: (i) Find the maximum UID among all agents which have communicated with it during the last δ units of time; (ii) If this UID is less than its own UID, then stay else move; (iii) If the decision is to move then choose one node from its children and parents randomly and move towards it.

TABLE IV

DEPTH-FIRST SEARCH

1: $l = \max\{j \mid j \in$ `buffer-uid`$_i\}$
2: **if** $l < i$ **then**
3: **return**: `stay`
4: **end if**
5: $\mathcal{M}_i(t)_k = \mathcal{M}_l(t-\tau_l)_k$ for $k \in \{1,3,4\}$
6: **if** $|\mathcal{M}_i(t)| = 2$ **then**
7: Compute $X = \mathcal{V}(p_i, Q)$
8: **else**
9: Compute $X = \mathcal{V}(p_i, Q_{p_i}(\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4))$
10: **end if**
11: Compute the list of gaps of X excluding $[\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4]$, say $\{[v'_{i_1}, v''_{i_1}], \ldots, [v'_{i_k}, v''_{i_k}]\}$ such that the list of vertices $\{p_i, v'_{i_1}, v''_{i_1}, \ldots, v'_{i_k}, v''_{i_k}\}$ is ordered counter-clockwise.
12: **if** $k = 0$ or ($\mathcal{M}_i(t)_2 \in [v'_{i_k}, v''_{i_k}]$ and $|\mathcal{M}_i(t)| > 2$) **then**
13: **return**: `to-parent`
14: **else**
15: $\mathcal{M}_i(t)_1 = p_i$
16: **if** $\mathcal{M}_i(t)_2 \in [\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4]$ **then**
17: $\mathcal{M}_i(t)_3 = v'_{i_1}$; $\mathcal{M}_i(t)_4 = v''_{i_1}$
18: **else**
19: **if** $\mathcal{M}_i(t)_2 \in [v'_{i_m}, v''_{i_m}]$ **then**
20: $\mathcal{M}_i(t)_3 = v'_{i_{m+1}}$; $\mathcal{M}_i(t)_4 = v''_{i_{m+1}}$
21: **end if**
22: **end if**
23: **return**: `to-child`
24: **end if**

TABLE V

RANDOMIZED SEARCH

1: $l = \max\{j \mid j \in$ `buffer-uid`$_i\}$
2: **if** $l < i$ **then**
3: **return**: `stay`
4: **end if**
5: $\mathcal{M}_i(t)_k = \mathcal{M}_l(t-\tau_l)_k$ for $k \in \{1,3,4\}$
6: **if** $|\mathcal{M}_i(t)| = 2$ **then**
7: Compute $X = \mathcal{V}(p_i, Q)$
8: **else**
9: Compute $X = \mathcal{V}(p_i, Q_{p_i}(\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4))$
10: **end if**
11: Compute the list of gaps of X say $\{[v'_{i_1}, v''_{i_1}], \ldots, [v'_{i_k}, v''_{i_k}]\}$ such that the list of vertices $\{p_i, v'_{i_1}, v''_{i_1}, \ldots, v'_{i_k}, v''_{i_k}\}$ is ordered counter-clockwise.
12: Generate a random number, say a (uniformly distributed over the interval $[0,1]$)
13: Let $a \in [\frac{m}{k}, \frac{m+1}{k})$ where $m \in \{0, \ldots, k-1\}$
14: **if** $[v'_{i_m}, v''_{i_m}] = [\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4]$ **then**
15: **return**: `to-parent`
16: **else**
17: $\mathcal{M}_i(t)_1 = p_i$
18: $\mathcal{M}_i(t)_3 = v'_{i_m}$; $\mathcal{M}_i(t)_4 = v''_{i_m}$
19: **return**: `to-child`
20: **end if**

A formal description of the randomized-search routine is given in Table V. In what follows we shall refer to the asynchronous algorithm in Table III together with the depth-first search routine in Table IV by $\mathcal{A}_{\text{dfs}}$. Also we shall use $\mathcal{A}_{\text{rs}}$ to refer to the asynchronous algorithm in Table III together with the depth-first search routine in Table V. The following lemma captures the fact that in the algorithms $\mathcal{A}_{\text{dfs}}$ and $\mathcal{A}_{\text{rs}}$, there is always enough information to successfully execute the depth-first and randomized searches.

Lemma 5.4: For any agent, i, let $p_i(t)$ represent the position of the agent at any time $t \in T^i$, say $t = T^i_l$. Then the following statements are true:

(i) $p_i(t) \in \mathcal{N}_Q(s)$, say $p_i(t) = \mathcal{N}_Q(s)_k$;
(ii) $\mathcal{M}_l(t-\tau_l)_1$ represents the location of the parent of $\mathcal{N}_Q(s)_k$, say $\mathcal{N}_Q(s)_j$, where $l = \max(\{j \mid j \in$ `buffer-uid`$_i\}, i)$;
(iii) $[\mathcal{M}_l(t-\tau_l)_3, \mathcal{M}_l(t-\tau_l)_4] = \mathcal{N}_Q(s)_k \cap \mathcal{N}_Q(s)_j$, where l is as defined above;
(iv) $\mathcal{M}_i(t)_2 \in \mathcal{N}_Q(s)_k \cap \mathcal{N}_Q(s)_j$ where $\mathcal{N}_Q(s)_j$ is the last node of $\mathcal{G}_Q(s)$ occupied by agent i.

2) Convergence analysis and time complexity: In this section, we analyze the convergence properties of the algorithms described in Section V-B. We also give an upper bound on the time complexity of the completion of the task. Before presenting these results, let us state an important observation.

Lemma 5.5: Given a simple polygonal environment Q without holes, consider a network of visually guided agents initially located at $s \in \text{Ve}(Q)$ exploring the vertex-induced tree $\mathcal{G}_Q(s)$ by executing either algorithm $\mathcal{A}_{\text{dfs}}$ or $\mathcal{A}_{\text{rs}}$. Then given any node $\mathcal{N}_Q(s)_k$ of $\mathcal{G}_Q(s)$, if at any time t there exists agent i such that $p_i(t) = \mathcal{N}_Q(s)_k$, then at any time $t' > t$, there exists agent j, such that $p_j(t') = \mathcal{N}_Q(s)_k$ with the property that $j \geq i$.

In other words, the number of occupied nodes is nondecreasing. We are now ready to state the main results of this paper.

Theorem 5.6: Given a simple polygon Q without holes, let $p_1(T^1_0) = \ldots = p_N(T^N_0) = s \in \text{Ve}(Q)$, represent the initial positions of an asynchronous network of N visually-guided agents as described in Section II. Let the behavior of the agents be governed by the algorithm $\mathcal{A}_{\text{dfs}}$. Then the following are true:

(i) there exists a finite time t^* after which there is at least one agent on $\min\{|\mathcal{N}_Q(s)|, N\}$ nodes of $\mathcal{G}_Q(s)$;
(ii) if $N \geq \lfloor \frac{n}{3} \rfloor$, then the visibility-based deployment problem is solved in finite time;
(iii) if there exist bounds $\lambda_{\max}$ and $\rho_{\max}$ such that $\lambda^i_l \leq \lambda_{\max}$ and $\rho^i_l \leq \rho_{\max}$ for all $i \in \{1, \ldots, N\}$ and $l \in \mathbb{N} \cup \{0\}$, then $t^* \leq \mathcal{T}_{\text{motion}} + \mathcal{T}_{\text{nodes}}$, where $\mathcal{T}_{\text{motion}} \leq 2\frac{\mathcal{L}_{\text{forward}}(\mathcal{G}_Q(s)) + \mathcal{L}_{\text{backward}}(\mathcal{G}_Q(s))}{\min\{\mathcal{L}_{\text{backward}}(\mathcal{G}_Q(s))_i \mid i \in \{1,\ldots,|\mathcal{N}_Q(s)-1|\}\}} - \frac{v}{v}$ and $\mathcal{T}_{\text{nodes}} \leq 2(\lambda_{\max} + \rho_{\max})(|\mathcal{N}_Q(s) - 1|)$, where v is the speed with which the agents move;

Proof: We first prove fact (i). Before beginning with the proof, let us define a *stationary agent*, i, to be any agent such that $p_i(t) = \mathcal{N}_Q(s)_k$ for all $t \geq t'$ where t' is some finite time instant and $k \in \{1, \ldots, |\mathcal{N}_Q(s)|\}$. It is easy to see that unless an agent is stationary, it performs a depth-first search on $\mathcal{G}_Q(s)$. Hence, an agent that is not stationary visits all the nodes of $\mathcal{G}_Q(s)$ in finite time. It is clear from steps (2)-(4) of the depth-first search routine, that no two agents at the same node can be stationary. Hence the number of such agents is less or equal to $\min\{|\mathcal{N}_Q(s)|, N\}$. If however, this number is strictly less than $\min\{|\mathcal{N}_Q(s)|, N\}$, it means that the number of stationary agents is strictly less than N. Hence at least one agent is not stationary. Also, it means that at least one node of $\mathcal{G}_Q(s)$ is not occupied by any agent. This is because if all nodes were occupied then from Lemma 5.5, there would always be an agent on every node and then in finite time there would be a stationary agent on that node. But since there is

one agent that is not stationary, it would reach the empty node of $\mathcal{G}_Q(s)$ in finite time through depth-first search. This is a contradiction.

Fact (i) and Lemma 4.5 together imply that if $N \geq \lfloor \frac{n}{3} \rfloor$, then at time t^*, the number of occupied nodes will be $|\mathcal{N}_Q(s)|$. Fact (ii) then follows trivially.

We do not include the proof of fact (iii) here in the interest of space. ■

Theorem 5.7: Given a simple polygon Q without holes, let $p_1(T_0^1) = \ldots = p_N(T_0^N) = s \in \mathrm{Ve}(Q)$, be the initial positions of an asynchronous network of N visually-guided agents as described in Section II. Then in finite time with high probability, the algorithm $\mathcal{A}_{\mathrm{rs}}$ solves the visibility-based deployment problem.

Proof: We only provide a sketch of the proof of the theorem here. The randomized search algorithm is similar to a random walk on a tree. An agent performing a random walk on a tree visits each of the nodes of the tree in finite time with high probability. Note that a random walk on a graph can also be modeled as a Markov chain. If there are now multiple agents performing random walks on the same tree independently, then again with high probability each of the agents is going to visit the all the nodes of the tree in finite time. Following on these lines, it can then be deduced that every empty node of vertex-induced tree is going to be visited by an agent in finite time with high probability. ■

C. Simulations

In this section we present simulation results for the algorithms described in the previous section. The algorithms have been implemented in MATLAB. The environment is chosen to represent a typical floor plan. See Figure 1 for the environment, Q, and the vertex-induced tree $\mathcal{G}_Q(s)$ where s is as shown in the figure.

Figures 7 and 8 show the results of the simulations of the algorithms $\mathcal{A}_{\mathrm{dfs}}$ and $\mathcal{A}_{\mathrm{rs}}$ respectively. The nodes of the vertex-induced tree of the environment in the simulations are precisely the locations where the agents in Figure 7 are located at the end of the simulation. In Figure 8, there are more agents than the number of nodes in the vertex-induced tree. Hence, the extra agents keep exploring the graph without coming to rest.

VI. Conclusions

In this paper, we provide a distributed solution to the visibility-based deployment problem. This problem is closely related to the classical Art Gallery problem. We also introduce a new graph to represent a given nonconvex environment without holes called the vertex-induced tree. We then demonstrate that with limited memory and based on information obtained through line-of-sight sensing and communication, multiple agents operating asynchronously can cover the nodes of this tree. The algorithms presented in this paper are guaranteed to solve the visibility-based deployment problem only if all the agents in the network are initially located at the same point. Possible extensions of this work include the design of algorithms that are guaranteed to work even if the agents do not start at the same location. Another direction is to investigate the algorithms for robustness to agent arrivals and departures.

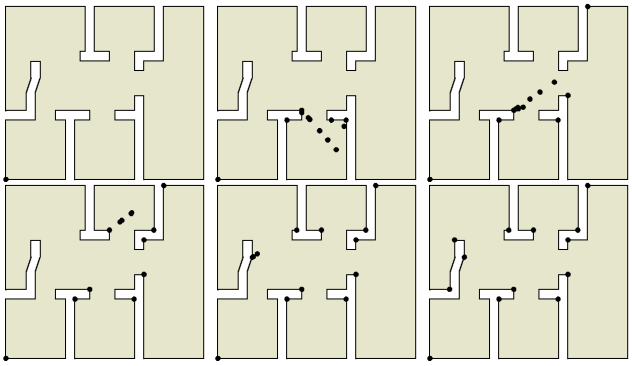

Fig. 7. From left to right and top to bottom, evolution of a network implementing the algorithm $\mathcal{A}_{\mathrm{dfs}}$; see Table IV. The number of vertices of the environment is $n = 46$ and the number of agents is $N = 13 < \lfloor \frac{46}{3} \rfloor$. Each point of the environment is visible at the end of the simulation.

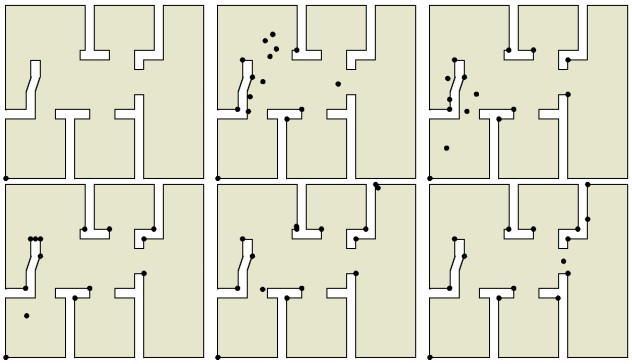

Fig. 8. From left to right and top to bottom, evolution of a network implementing the algorithm $\mathcal{A}_{\mathrm{rs}}$; see Table V. The number of vertices of the environment is $n = 46$ and the number of agents $N = 15 < \lfloor \frac{46}{3} \rfloor$. The vertex-induced tree has 13 nodes, so the 2 extra agents continue to explore the vertex-induced tree. Each point of the environment is visible at the end of the simulation.

References

[1] T. C. Shermer, "Recent results in art galleries," *IEEE Proceedings*, vol. 80, no. 9, pp. 1384–1399, 1992.

[2] V. Chvátal, "A combinatorial theorem in plane geometry," *Journal of Combinatorial Theory. Series B*, vol. 18, pp. 39–41, 1975.

[3] S. Fisk, "A short proof of Chvátal's watchman theorem," *Journal of Combinatorial Theory. Series B*, vol. 24, p. 374, 1978.

[4] A. Howard, M. J. Matarić, and G. S. Sukhatme, "An incremental self-deployment algorithm for mobile sensor networks," *Autonomous Robots*, vol. 13, no. 2, pp. 113–126, 2002.

[5] J. Grace and J. Baillieul, "Stochastic algorithms for autonomous robotic surveillance," in *IEEE Conf. on Decision and Control*, Seville, Spain, Dec. 2005, to appear.

[6] L. Guilamo, B. Tovar, and S. M. LaValle, "Pursuit-evasion in an unkown environment using gap navigation trees," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, Sendai, Japan, Sept. 2004, pp. 3456–3462.

[7] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont, MA: Athena Scientific, 1997.