# 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 simple nonconvex polygonal environment. 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 deploy over the nodes of this tree by means of distributed algorithms. The agents are assumed to have access to a local memory and their operation is partially asynchronous.**

## I. Introduction

Imagine an art gallery whose floor plan can be modeled as a nonconvex polygon. Now consider the problem of finding the least number of stationary guards so that each point of the art gallery is visible to at least one guard. A stationary guard, here, is a fixed point that can see in every direction or equivalently, has omnidirectional vision. The assumption here is that guards cannot see through the walls of the environment. This is the statement of the classical Art Gallery Problem. We can also think of this problem as that of illuminating a polygonal environment with point lights; see Fig. 1 for a graphical illustration of the objective.

Inspired by this and other ”illumination problem” (see the beautiful survey [1]), we pose the following problem. Imagine a group of mobile robots, modeled as point masses, in a nonconvex polygonal environment. Each robot has omnidirectional vision and also has line-of-sight wireless communication capabilities. The problem then is to design a distributed algorithm, copies of which run will run on each robot and drive them to locations such that each point of the environment is visible to at least one robot. The algorithm is distributed in the sense that it depends only on information obtained from local sensing and communication. We also assume that the robots operate asynchronously. In what follows, we shall refer to this problem as the *visibility-based deployment problem*.

This problem is related to many surveillance and pursuit-evasion problems in unknown environments. Some related works include [2] where an incremental heuristic for deployment is proposed (no formal analysis is presented) and [3] in which the relevance of random walk on graphs is discussed (the environment and its graphical representation are assumed known a priori, general strategies are evaluated via Monte Carlo simulation). In addition, the proposed work is related to visibility-based pursuit-evasion problems, see [4], [5], although these works focus on single agents and not on distributed policies for groups of agents.

Clearly, the solution to the visibility-based deployment problem exists only if sufficient number of robots are present to complete the task. According to the famous Art Gallery Theorem [6], $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and occasionally necessary to guard a simply connected polygon with $n$ vertices. Fisk's constructive proof of this theorem [7] provides an elegant way of finding the guard locations. However, the construction relies on complete knowledge of the environment and is contrary to the assumptions in our formulation of the problem. As we shall see later, the conservativeness of this assumption leads us to an algorithm that in the worst case requires $\lfloor \frac{n}{2} \rfloor$ robots as against $\lfloor \frac{n}{3} \rfloor$.

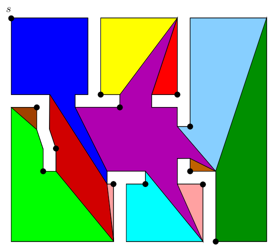


Fig. 1. A nonconvex polygon shaped like a typical floor plan: the solid circles represent the locations of *guards* with omnidirectional vision. Note that each portion of the environment is visible to at least one guard. Every colored subset is a star-shaped polygon visible from the guard located in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

We now present a summary of our approach and contribution. In what follows we shall use the term agents to refer to robots or guards. First, given a simple nonconvex polygonal environment and one of its vertices, we describe a procedure to incrementally partition the environment into star-shaped polygons. This induces a new graph, associated to the nonconvex polygon and to the given vertex, called the vertex-induced tree; each star-shaped polygon is a node and an edge exists between two nodes if and only if the corresponding star-shaped polygons share an edge. Second, we design local navigation algorithms to move between neighboring nodes of the vertex-induced tree. Third, we present asynchronous and distributed global algorithms for multiple agents to deploy over the nodes of the vertex-induced tree, and thereby solve the visibility-based deployment problem. The algorithms are based on information obtained from local sensing and communication, and also on some limited local memory.

*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 $\overline{B}_r(p)$ 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

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

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 polygon, possibly nonconvex. A polygon is said to be simple if it does not contain any hole. Let $\mathrm{Ve}(Q) = (v_1, \dots, v_n)$ be the list of vertices of $Q$ ordered counterclockwise. A reflex vertex is a vertex of $Q$ where the internal angle is greater than $pi$ radians. 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 $p$. Also, we shall use $P$ to refer to tuples of elements in $\mathbb{R}^2$ of the form $(p_1, \dots, p_N)$. With a slight abuse of notation, we shall use $P$ interchangeably with a point set of the form $\{p_1, \dots, p_N\}$.

## II. Network modeling and problem description

In this section, we describe in detail the sensing and communication capabilities that define a visually-guided agent. Each agent has a unique identifier (UID), $i \in \{1, \dots, N\}$. Furthermore, it is equipped with an omnidirectional line-of-sight range sensor. Thus, agent $i$ located at $p_i$ can measure the relative position of any other agent or of a point on the environment boundary that lies in its visibility polygon $S(p_i)$. An agent can also communicate with any other agent visible to it and at a distance less than the communication radius $r_i > 0$. The communication is assumed to be UDP based. Thus, the communication region can be denoted by $\mathcal{C}(p_i) = S(p_i) \cap \overline{B}_{r_i}(p_i)$. We also assume here that $r_i$ can be chosen freely by an agent but cannot exceed a certain upper bound, say $R$. Whenever agent $i$communicates, it sends a BROADCAST$_i(i, \mathcal{M}_i)$ message containing its UID $i$ and the contents of its memory $\mathcal{M}_i$. We assume here that there is a variable but bounded time delay between sending and receiving of messages. Let $\delta > 0$ denote this upper bound.

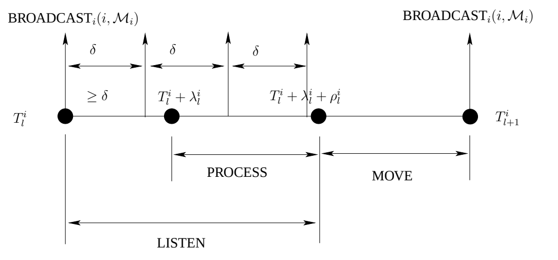
To any agent $i$, we associate a sequence $(T^i)_k$ of wake-up instants. Let $T_0^1 = \dots, T_0^N = t_0$. The agent performs the following actions between $T_l^i$ and $T_{l+1}^i$ for any $l$:

(i) BROADCAST$_i(i, \mathcal{M}_i)$ at times $t = T_l^i + k\delta$, where $k \in \mathbb{N} \cup \{0\}$, as long as the agent is not moving;
(ii) LISTEN for broadcasts during the time interval $[T_l^i, T_l^i + \lambda_l^i)$, $\lambda_l^i \geq \delta$;
(iii) Continue to LISTEN and PROCESS states in its memory during the interval $[T_l^i + \lambda_l^i, T_l^i + \lambda_l^i + \rho_l^i)$;
(iv) MOVE during the time interval $[T_l^i + \lambda_l^i + \rho_l^i, T_{l+1}^i)$. If the robot decides not to move then $T_{l+1}^i = T_l^i + \lambda_l^i + \rho_l^i$.

*Remarks 2.1:* (i) Note that the sequence $T^i$ is not prespecified. Given any wake-up instant $T_l^i$, the next wake-up instant $T_{l+1}^i$ 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 Fig. 2 for a schematic illustration of the above schedule.

Agent $i$, in the MOVE state, is capable of moving at any time $t \in [T_l^i + \lambda_l^i + \rho_l^i, T_{l+1}^i)$ according to the following



BROADCAST$_i(i, \mathcal{M}_i)$
BROADCAST$_i(i, \mathcal{M}_i)$
$\delta$
$\delta$
$\delta$
$\geq \delta$
$T_l^i + \lambda_l^i$
$T_l^i + \lambda_l^i + \rho_l^i$
$T_l^i$
$T_{l+1}^i$
PROCESS
MOVE
LISTEN

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.

discrete-time control system:

$$p_i(t + \Delta t) = p_i(t) + u_i, \quad (1)$$

where the control is a function of the action that the agent performs at time $t$, the memory $\mathcal{M}_i(t)$, and the information obtained from communication and sensing. Also, note that this model of visually-guided agents is similar in spirit to the *partially asynchronous model* described in [8].

Having described the model, we are now in a position to formally describe the visibility-based deployment problem.

> Given a polygonal environment $Q$, let $p_1(t_0), \dots, p_N(t_0) \in Q$ represent the initial positions of an asynchronous network of $N$ visually-guided agents as described in Section II. The visibility-based deployment problem is solved if the agent dynamics dictated by a suitable control law as described in (1) causes the positions of the agents to converge to a set $W \subset (2^Q)^N$ with the property that $\cup_i^N S(p_i) = Q$ for all $(p_1, \dots, p_N) \in W$.

## III. The vertex-induced tree

Let us start by describing a procedure to partition[*] a simple polygonal environment, $Q$, into star-shaped subsets. To begin constructing this partition, we require a starting vertex which we call $s \in \mathrm{Ve}(Q)$. The procedure is as follows:

```
Let $v = s$ and $X = Q$
1: Compute the set of all vertices of $X$ visible from $v$
2: Let $\mathcal{P}_{\text{temp}}$ be the polygon defined by the set of vertices
   and insert it into the list $\mathcal{P}$
3: Insert $v$ into another list $\mathcal{N}$
4: Find all edges of $\mathcal{P}_{\text{temp}}$ that are diagonals of $X$ and
   insert them into list $G$. We call these diagonals as *gaps*.
5: **for** all gaps in $G$ **do**
6:    Find a vertex $v'$ across the gap at a minimum
      distance from the mid-point of the diagonal and from
      which the complete gap is visible
7:    Perform steps 1- 4 (with $v = v'$ and $X$ being the
      environment across the gap)
8: **end for**
```

See Fig. 3 (left) for a graphical description of the algorithm.

[*]Recall that a partition of any set $X$ is a collection of closed and connected subsets $X_i$ with mutually disjoint interiors such that $X = \cup_i X_i$.

In the procedure just described, $\mathcal{P}$ is the list comprising of star-shaped polygons composing the partition and $\mathcal{N}$ as the list of kernel[†] points of the star-shaped polygons. In other words, if $\mathcal{P}_i$ be the $i$th element of $\mathcal{P}$, all points of $\mathcal{P}_i$ are visible from the vertex $\mathcal{N}_i$. Henceforth, given $Q$ and a vertex $s \in Q$, we shall refer to this partition as $\mathcal{P}_Q(s)$ and to the list $\mathcal{N}$ as $\mathcal{N}_Q(s)$. Finally, we refer to $\mathcal{P}_Q(s)$ as the *vertex-induced partition*.

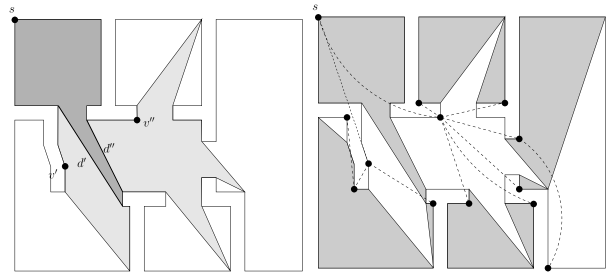


Fig. 3. The figure on the left shows the incremental way in which the vertex-induced partition is constructed. Starting from the vertex $s$, the polygon defined by all the vertices visible from $s$ is constructed. This is represented by the dark shaded polygon. The thick edges of this polygon are diagonals, $d'$ and $d''$, of the environment. $v'$ and $v''$ are vertices of the environment on the other side of the diagonals at a minimum distance from the mid-point of the diagonals. The lighter shaded polygons are the sets of vertices visible from $v'$ and $v''$ on the other side of $d'$ and $d''$. This procedure is repeated until the entire environment is partitioned. The figure on the right shows the partition. The vertex-induced tree is also shown. 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 $s$.

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 3.1:* 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) 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 standard notions of graph theory.

*Definition 3.2:* Given a simple polygon $Q$ 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 $\mathcal{G}_Q(s)$. Some important properties of the vertex-induced tree are as follows.

*Lemma 3.3:* Given a simple polygon $Q$ 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) $|\mathcal{N}_Q(s)| \leq \frac{n}{2}$ where $n = |\mathrm{Ve}(Q)|$.

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

## IV. Deployment 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. These algorithms are a result of local navigation algorithms and global deployment schemes.

### A. Local 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 4.1:* Given a simple polygon $Q$ without holes and any vertex $s \in \mathrm{Ve}(q)$, let $\mathcal{N}_Q(s)_i, \mathcal{N}_Q(s)_j$ represent two neighboring nodes of the vertex-induced tree $\mathcal{G}_Q(s)$. Let $[v', v''] = \mathcal{P}_Q(s)_i \cap \mathcal{P}_Q(s)_j$ 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. The following is an informal description of the MOVE-TO-PARENT routine to travel from a node to its parent:

```
MOVE-TO-PARENT
1: compute the shortest path between the node and the
   parent based on Lemma 4.1
2: go to the reflex vertex which is a part of the shortest
   path
3: from the nonconvex vertex, go to the vertex represent-
   ing the parent node
```

Next, we present an informal description of the MOVE-TO-CHILD routine to travel from a node to a child.

```
MOVE-TO-CHILD
1: compute the mid-point of the gap between the node
   and the child
2: go to the mid-point
3: compute the nearest vertex from which the entire gap
   is visible and which is across the gap
4: go to that vertex
```

The formal descriptions of these routines appear in Tables II and III in the Appendix. See Fig. 4 for a graphical illustration of the paths between nodes and the respective parents and children.

Note that the algorithms described in this section require the knowledge of the relative locations of the parent $p_{\text{parent}}$ and the vertices $v', v''$ defining the gap between the node and its parent, or the gap between the node and a child. All this information is obtained using local sensing and communication. This distributed information processing capability is in-built into the global deployment schemes described in the following section.

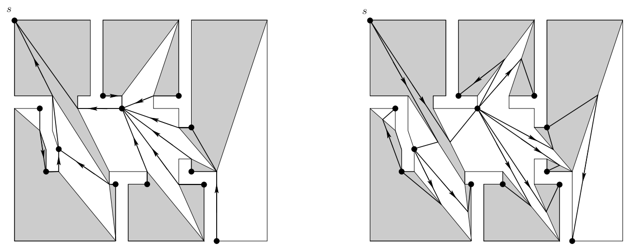


Fig. 4. The figure on the left shows the planned paths from nodes to their parent in the vertex-induced tree. The figure on the right shows the planned paths from nodes to their respective children in the vertex-induced tree.

### B. Global deployment schemes

Note that by virtue of the construction in Section III 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 problem of deployment over a graph.

*1) Deploying over the vertex-induced tree:* In this section, we design algorithms for multiple agents to deploy over 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 deployment algorithms. It must be noted that these algorithms may not be optimal in terms of performance measures such as required time. Our main aim is to provide a solution to the visibility-based deployment problem. Performance issues will be the subject of future research.

Let us first informally describe the deployment algorithms.

> Each agent repeatedly performs the following tasks whenever it is located at a node of the vertex-induced tree:
> (i) Find the maximum UID among all agents located at the same node; (ii) If this UID is less than its own UID, then stay else move; (iii) If the decision is to move then decide upon the next node to be visited and move to it.

To decide upon the next node to be visited, two different methods are described: (i) depth-first deployment, and (ii) randomized deployment. The essential difference between the two methods is that in the depth-first deployment, the next node to be visited is decided in a deterministic way while in the randomized deployment the decision is random. We formally describe the depth-first and randomized deployment routines in Table IV in the Appendix.

To ensure that sufficient information is present to implement the algorithms, we propose the following communication region and memory for agent $i$. See also Lemmas 4.2 and 4.3.

(i) The communication region specified by $\mathcal{C}(p_i) = S(p_i) \cap \overline{B}_{p_i}(r)$, where $r = \min\{R, \frac{1}{2}\min\{\|p_i - v\|,\ v \in \mathrm{Ve}(S(p_i))\}\}$, if $p_i \in \mathrm{Ve}(Q)$. This ensures that communication broadcasts reach only agents located at the same node as $i$.

(ii) The memory $\mathcal{M}_i(t) = \{p_{\text{parent}}, p_{\text{last}}, v', v''\}$ comprising of four points in $Q$. The four points refer to the relative locations of the parent node, the last way point (a node or mid-point of a gap), the two vertices defining the gap between the current node and the parent. We let $\mathcal{M}_i(t)_k$ refer to the $k$th element of the list.

(iii) The list `buffer-uid`$_i$ whose elements are natural numbers.

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

We are now in a position to formally describe the various actions performed by an agent in between two wake-up instants; see Table I. Note that the depth-first and randomized deployment routines are invoked during PROCESS.

TABLE I
DESCRIPTION OF VARIOUS ACTIONS TO DEPLOY OVER NODES OF $\mathcal{G}_Q(s)$

```
Assumes:   $p_1(t_0) = \ldots = p_N(t_0) = s \in \mathrm{Ve}(Q)$

0: Assume $k$ s.t. $\mathcal{N}_k = p_i$
0: buffer-uid$_i = \emptyset$; buffer-memory$_i = \emptyset$
0: $\mathcal{M}_i(t_0) = \{p_i(t_0), p_i(t_0)\}$
0: move-decision := stay
Any agent $i$, executes the following actions according to the schedule in
Section II at any time $t$ 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 $0 \le \tau \le \delta$
 2: if $j$ already exists in buffer-uid$_i$ then
 3:    Swap older $\mathcal{M}_j(t')$ in buffer-memory$_i$ with newer $\mathcal{M}_j(t-\tau)$
 4: else
 5:    Append $j$ to buffer-uid$_i$
 6:    Append $\mathcal{M}_j(t-\tau)$ to buffer-memory$_i$
 7: end if
PROCESS
 1: run Depth-first or Randomized deployment
MOVE
 1: switch move-decision
 2:    case stay: Stay at $\mathcal{N}_k$
 3:    case to-child: buffer-uid$_i = \emptyset$; buffer-message$_i = \emptyset$;
       run MOVE-TO-CHILD($\mathcal{M}_i(t)$)
 4:    case to-parent: buffer-uid$_i = \emptyset$; buffer-message$_i =$
       $\emptyset$; run MOVE-TO-PARENT($\mathcal{M}_i(t)$)
 5: end switch
```

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

*Lemma 4.2:* For any agent $i$ at any time $t$, if `buffer-uid`$_i \neq \emptyset$, then $p_i(t) \in \mathcal{N}_Q(s)$ and there exists $\tau_j$ with $0 \le \tau_j \le \delta$ such that $p_j(t-\tau_j) = p_i(t)$ for all $j \in$ `buffer-uid`$_i$.

In what follows we shall refer to the depth-first deployment routine together with the local navigation algorithms by $\mathcal{A}_{\text{dfd}}$. Similarly we shall use $\mathcal{A}_{\text{rd}}$ to refer to the randomized deployment routine. The following lemma captures the fact that in the algorithms $\mathcal{A}_{\text{dfd}}$ and $\mathcal{A}_{\text{rd}}$, there is always enough information to successfully execute the depth-first and randomized deployment algorithms.

*Lemma 4.3:* 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:* In this section, we analyze the convergence properties of the algorithms described in Section IV-B. We also give an upper bound on the time to completion of the task.

*Theorem 4.4 (Depth-first deployment):* Given a simple polygon $Q$, let $p_1(t_0) = \ldots = p_N(t_0) = s \in \mathrm{Ve}(Q)$, be 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 $\mathcal{A}_{\text{dfd}}$. Then the following are true:

(i) there exists a finite time $t^*_{\text{dfd}}$ after which there is at least one agent on $\min\{|\mathcal{N}_Q(s)|, N\}$ nodes of $\mathcal{G}_Q(s)$;
(ii) if $N \geq \frac{n}{2}$, then the visibility-based deployment problem is solved in finite time.

We now present a run-time analysis of $\mathcal{A}_{\text{dfd}}$. But before that, let us introduce some notation regarding the lengths of paths between two nodes of the vertex-induced tree. Note from Fig. 4 the path from a node to its parent is shorter than the path from the parent to the node.

*Definition 4.5:* Given a simple polygonal environment $Q$, we define the following:

(i) $\mathcal{L}_{\text{fwd}}(\mathcal{G}_Q(s))_i$, the length of the path from a node to the child which are part of the edge $i$;
(ii) $\mathcal{L}_{\text{bwd}}(\mathcal{G}_Q(s))_i$, the length of the path from a node to its parent which are part of the edge $i$;
(iii) the forward length of the graph $\mathcal{G}_Q(s)$, $\mathcal{L}_{\text{fwd}}(\mathcal{G}_Q(s)) = \sum_{i=1}^{|\mathcal{N}_Q(s)|-1} \mathcal{L}_{\text{fwd}}(\mathcal{G}_Q(s))_i$;
(iv) the backward length of the graph $\mathcal{G}_Q(s)$ $\mathcal{L}_{\text{bwd}}(\mathcal{G}_Q(s)) = \sum_{i=1}^{|\mathcal{N}_Q(s)|-1} \mathcal{L}_{\text{bwd}}(\mathcal{G}_Q(s))_i$.

*Proposition 4.6 (Run-time analysis):* 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 $t^*_{\text{dfd}} \leq \mathcal{T}_{\text{motion}} + \mathcal{T}_{\text{comm/sens/proc}}$; where

$$\mathcal{T}_{\text{motion}} \leq \frac{1}{v}\Big(2\big(\mathcal{L}_{\text{fwd}}(\mathcal{G}_Q(s)) + \mathcal{L}_{\text{bwd}}(\mathcal{G}_Q(s))\big) - \min\{\mathcal{L}_{\text{fwd}}(\mathcal{G}_Q(s))_i \mid i \in \{1, \ldots, |\mathcal{N}_Q(s)| - 1\}\}\Big),$$
$$\mathcal{T}_{\text{comm/sens/proc}} \leq 2(\lambda_{\max} + \rho_{\max})\,(|\mathcal{N}_Q(s)| - 1)\,,$$

and $v$ is the speed with which the agents move. Moreover, as $N$ and $|\mathrm{Ve}(Q)| \to +\infty$, if the diameter of $Q$ is bounded, then $t^*_{\text{dfd}} \in \Theta(\min\{N, |\mathrm{Ve}(Q)|\})$.
In other words, in the worst case, the run-time is uniformly upper and lower bounded by constant multiples of $\min\{N, |\mathrm{Ve}(Q)|\}$.

*Theorem 4.7 (Randomized deployment):* Given a simple polygon $Q$, let $p_1(t_0) = \ldots = p_N(t_0) = s \in \mathrm{Ve}(Q)$, be 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 $\mathcal{A}_{\text{rd}}$. Then the following are true:

(i) with high probability in finite time, there is at least one agent on $\min\{|\mathcal{N}_Q(s)|, N\}$ nodes of $\mathcal{G}_Q(s)$;
(ii) if $N \geq \frac{n}{2}$, then the visibility-based deployment problem is solved in finite time with high probability.

## C. Simulations

In this section we present simulation results for the algorithms, $\mathcal{A}_{\text{dfd}}$ and $\mathcal{A}_{\text{rd}}$, described earlier. The algorithms have been implemented in `MATLAB`. The environment, $Q$, the root $s$ and the vertex-induced tree $\mathcal{G}_Q(s)$ are as shown in Fig. 3 (right). Note that $Q$ is chosen to represent a typical floor plan. Figs. 5 and 6 show results for the algorithms $\mathcal{A}_{\text{dfd}}$ and $\mathcal{A}_{\text{rd}}$ respectively. The nodes of the vertex-induced tree of the environment in the simulations are precisely the locations where the agents in Fig. 5 are located at the end of the simulation. In Fig. 6, 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.

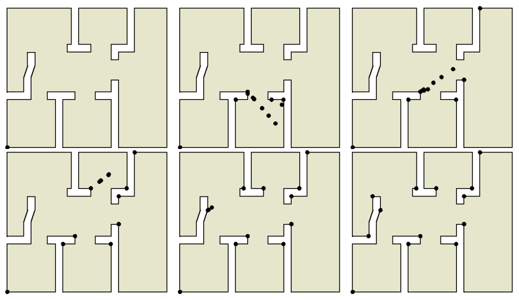


Fig. 5. From left to right and top to bottom, evolution of a network implementing the algorithm $\mathcal{A}_{\text{dfd}}$; see Table IV. The number of vertices of the environment is $n = 46$ and the number of agents is $N = 13 < \lfloor \frac{46}{3} \rfloor$. Each point of the environment is visible at the end of the simulation.

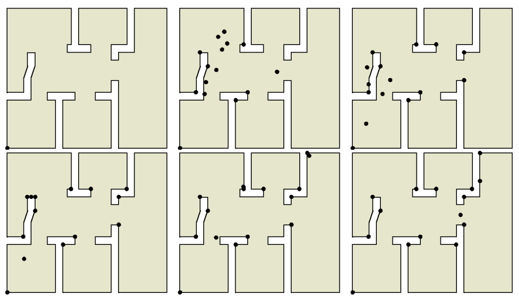


Fig. 6. From left to right and top to bottom, evolution of a network implementing the algorithm $\mathcal{A}_{\text{rd}}$; see Table IV. The number of vertices of the environment is $n = 46$ and the number of agents $N = 15 < \lfloor \frac{46}{3} \rfloor$. The vertex-induced tree has 13 nodes, so the 2 extra agents continue to explore the vertex-induced tree. Each point of the environment is visible at the end of the simulation.

## V. Conclusions

In this paper, we introduce the visibility-based deployment problem and provide a solution to it under the assumption that all agents are initially collocated. This problem is closely related to the classical Art Gallery Problem. We introduce a new graph to represent a given simple polygonal environment 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 deploy over the nodes of this tree. Note that once the visibility-based deployment problem is solved and visibility information from all the nodes is fused, the task of building a map of the environment or planning a path between two points of the environment becomes trivial. Other 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 failures.

## VI. Acknowledgment

This material is based upon work supported in part by AFOSR through Award F49620-02-1-0325, by ONR through YIP Award N00014-03-1-0512, and by NSF through CAREER Award ECS-0546871.

## References

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

[2] 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.

[3] J. Grace and J. Baillieul, "Stochastic algorithms for autonomous robotic surveillance," in *IEEE Conf. on Decision and Control and European Control Conference*, Seville, Spain, Dec. 2005, pp. 2200–2205.

[4] 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.

[5] V. Isler, S. Kannan, and S. Khanna, "Randomized pursuit-evasion in a polygonal environment," *IEEE Transactions on Robotics*, vol. 5, no. 21, pp. 864–875, 2005.

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

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

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

## VII. Appendix

### A. Local navigation algorithms

The MOVE-TO-PARENT and MOVE-TO-CHILD algorithms are described in Tables II and III respectively.

TABLE II
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 $\mathcal{N}_Q(s)_j$
Assumes:  (i)$[v', v''] = \mathcal{P}_Q(s)_i \cap \mathcal{P}_Q(s)_j$,
          (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: $\{p_{\text{parent}}, p_{\text{last}}, v', v''\}$
```

### B. Depth-first and randomized deployment algorithms

TABLE III
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 $\mathcal{N}_Q(s)_j$
Assumes:  $[v', v''] = \mathcal{P}_Q(s)_i \cap \mathcal{P}_Q(s)_j$

 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 $p \neq p_{\text{temp}}$ OR $p_{\text{last}} \neq \frac{v'+v''}{2}$ do
 5:   if $p_{\text{temp}} = \frac{v'+v''}{2}$ 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:       $p_{\text{temp}} = \arg\min\{\|v - (v' + v'')/2\| \mid [v', v'']$ is visible from $v$, $v$ is a vertex of $Q$
          across the gap $[v', v'']$ from $p_{\text{last}}\}$
 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: $\{p_{\text{parent}}, p_{\text{last}}, v', v''\}$
```

TABLE IV
DEPLOYMENT ALGORITHMS

```
 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 polygon $X$ defined by the set of vertices of $Q$ visible
      from $p_i$
 8: else
 9:   Compute polygon $X$ defined by the set of vertices of $Q$ visible
      from $p_i$ which are not across the gap $[\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4]$
10: end if
11: Compute the list of gaps of $X$ excluding $[\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4]$,
    say $\{[v'_{i_1}, v''_{i_1}], \dots, [v'_{i_k}, v''_{i_k}]\}$ such that the list of vertices
    $\{p_i, v'_{i_1}, v''_{i_1}, \dots, v'_{i_k}, v''_{i_k}\}$ is ordered counter-clockwise.
```

Depth-first deployment

```
 1: if $k = 0$ or ($\mathcal{M}_i(t)_2 \in [v'_{i_k}, v''_{i_k}]$ and $|\mathcal{M}_i(t)| > 2$) then
 2:   return: to-parent
 3: else
 4:   $\mathcal{M}_i(t)_1 = p_i$
 5:   if $\mathcal{M}_i(t)_2 \in [\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4]$ then
 6:     $\mathcal{M}_i(t)_3 = v'_{i_1}$; $\mathcal{M}_i(t)_4 = v''_{i_1}$
 7:   else
 8:     if $\mathcal{M}_i(t)_2 \in [v'_{i_m}, v''_{i_m}]$ then
 9:       $\mathcal{M}_i(t)_3 = v'_{i_{m+1}}$; $\mathcal{M}_i(t)_4 = v''_{i_{m+1}}$
10:     end if
11:   end if
12:   return: to-child
13: end if
```

Randomized deployment

```
 1: Generate a random number, say $a$ (uniformly distributed
    over the interval $[0, 1]$)
 2: Let $a \in [\frac{m}{k}, \frac{m+1}{k})$ where $m \in \{0, \dots, k-1\}$
 3: if $[v'_{i_m}, v''_{i_m}] = [\mathcal{M}_i(t)_3, \mathcal{M}_i(t)_4]$ then
 4:   return: to-parent
 5: else
 6:   $\mathcal{M}_i(t)_1 = p_i$
 7:   $\mathcal{M}_i(t)_3 = v'_{i_m}$; $\mathcal{M}_i(t)_4 = v''_{i_m}$
 8:   return: to-child
 9: end if
```