

# Distributed Pursuit-Evasion without Global Localization via Local Frontiers

Joseph W. Durham · Antonio Franchi · Francesco Bullo

Received: date / Accepted: date

**Abstract** This paper addresses a visibility-based pursuit-evasion problem in which a team of mobile robots with limited sensing and communication capabilities must coordinate to detect any evaders in an unknown, multiply-connected planar environment. Our distributed algorithm can guarantee evader detection and is built around maintaining complete coverage of the frontier between cleared and contaminated regions while expanding the cleared region. We detail a novel distributed method for storing and updating this frontier without building a global map or requiring global localization. We demonstrate the functionality of the algorithm through Player/Stage simulations in realistic environments and through hardware experiments. We also discuss the number of agents required to clear a certain class of environments, as well as compare Monte Carlo results for our algorithm to the theoretical optimum area cleared with a fixed number of robots.

## 1 Introduction

This paper deals with a distributed pursuit-evasion problem for a team of robotic searchers in an unknown environment. The particular *pursuit-evasion problem*

we examine, also known as the *clearing problem*, involves designing control and communication protocols such that the searchers sweep an environment and detect any intruders which may be present. The clearing problem has received a lot of attention in recent years because of its applications to safety and security. In this paper, we describe a distributed environment clearing algorithm based on the concept of the *frontier* or boundary between cleared and contaminated regions. Our algorithm can guarantee the detection of any intruders or, if there are insufficient searchers available, clear as much area as it can while ensuring no cleared area is recontaminated.

### 1.1 Literature Review

In the literature on pursuit-evasion problems, many different approaches and starting assumptions have been explored. The study of guaranteeing detection of evaders in planar environments began with Suzuki and Yamashita (1992). For a single searcher, Gerkey et al (2006) studied a searcher with a limited field of view in a known polygon, while Sachs et al (2004) cleared unknown environments without localization using minimalist sensing. The most similar work to this one is Kolling and Carpin (2010), which uses coordinated sweep lines of agents to clear unknown environments while building a graph representing the cleared space.

Pursuit-evasion on graphs representing decompositions of known environments is a related topic which goes back to Parsons (1978) and includes recent works by Adler et al (2003) and Kolling and Carpin (2008). Another active area is efficient evader detection, where one or more searchers are tasked with probabilistically locating targets which move randomly (Hollinger et al,

---

J. W. Durham and F. Bullo  
Department of Mechanical Engineering  
University of California  
Santa Barbara, CA 93106, USA  
E-mail: (joey, bullo)@engineering.ucsb.edu

A. Franchi  
Max Planck Institute for Biological Cybernetics  
Dept. Human Perception Cognition and Action  
Spemannstrasse 44, 72076 Tübingen, Germany  
E-mail: antonio.franchi@tuebingen.mpg.de

2010). The pursuit-evasion literature has also addressed what to do once evaders are located, including tracking moving evaders (Jung and Sukhatme, 2002) and capturing evaders (Bopardikar et al, 2008).

Beyond pursuit-evasion, our work draws inspiration from methods for exploration and deployment of agents based on the frontier between explored and unknown regions (Yamauchi, 1998; Howard et al, 2002; Franchi et al, 2009a).

## 1.2 Statement of Contributions

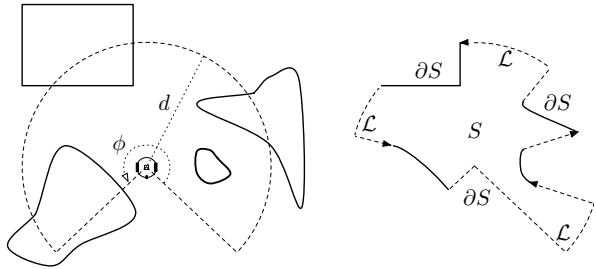
There are three key contributions of this work. First, our frontier-based clearing algorithm can guarantee detection of evaders in unknown, multiply-connected planar environments which may be non-polygonal. We introduce the  $(d, \phi)$ -searcher model, a realistic model of current robot and sensor hardware with limited range and limited field-of-view sensing, and prove that our algorithm will clear an environment provided sufficient searchers are available. We also discuss the number of agents required for certain environments.

Second, our clearing algorithm is distributed and efficient. We detail a novel method for storing and updating the global frontier between cleared and contaminated areas based on local intersections of oriented arcs. This method uses a small, constant amount of memory per robot and does not require a global map or global localization. We also propose a viewpoint planning method which locally minimizes the number of robots required to rapidly expand the cleared area.

Third, we present both realistic simulations and hardware experiments to validate our approach. We implemented the algorithm using the Multirobot Integration Platform and the Player/Stage robot simulation system. Our implementation demonstrates that frontiers and sensor footprints can be handled in a discretized fashion, that the algorithm is robust to sensor and motion noise, and that the local optimizations in our algorithm lead to efficient clearing of complex environments. We also present Monte Carlo results for the clearing efficacy of our algorithm for a fixed number of robots.

## 1.3 Paper Organization

Section 2 provides definitions and states the problem we are addressing. In 3 we examine a centralized version of our algorithm to explain some details. The decentralized clearing algorithm is presented in 4 and then demonstrated through simulations and experiments. In



**Fig. 1** On the left, four obstacles surround a  $(d, \phi)$ -searcher and lie within the dashed circular sector representing the area perceivable by the searcher’s sensor without occlusions. The right image shows the boundary  $\partial S$  of the sensor footprint for this configuration, with dashed oriented arcs for the free boundary  $\mathcal{L}$  and solid arcs for the local obstacle boundary

5 and 6 we discuss theoretical and numerical results. We conclude in 7 and mention some future directions.

## 2 Searcher Model & Problem Formulation

We are given a team of  $n$  robotic searchers with limited sensing and communication capabilities and finite memory. The searchers start clustered together in the free space of an unknown but limited planar environment. Let  $Q$  be the free space of the environment, which must be connected but can have holes and may be non-polygonal. The searchers are tasked with detecting evaders which may be arbitrarily small (even a single point) and can move arbitrarily fast, but continuously, through  $Q$ . The trajectories and initial positions of the evaders are unknown.

### 2.1 Robot and Sensor Models

The robot model we use, the  $(d, \phi)$ -searcher, is a differential or omnidirectional drive mobile robot that can rotate in place and translate continuously at bounded speed through  $Q$ . Our model gets its name from the attached distance sensor which has a maximum range  $d > 0$  and an angular field-of-view  $\phi \in [\pi, 2\pi]$ . The sensor cannot penetrate obstacles but is capable of detecting any evaders visible to it. We will also discuss the  $d$ -searcher model, which is a  $(d, \phi)$ -searcher with  $\phi = 2\pi$ .

Let  $S$  denote the *footprint* of the sensor when a robot is in a generic configuration, as shown in Fig. 1. The footprint is a local obstacle free region and we say that a point is *guarded* by a robot if it belongs to the footprint of the sensor of that robot. The oriented *boundary* of the sensor footprint,  $\partial S$  of  $S$ , is a closed arc

partitioned into two sets: (1) the *local obstacle boundary* (all the points where the sensor has perceived an obstacle), and (2) the *free boundary*, denoted by  $\mathcal{L}$ , which consists of all the remaining points. Notice that while  $S$  is always a simply connected region,  $\mathcal{L}$  is not, in general, a connected set. We refer to the connected subsets of  $\mathcal{L}$  as *free arcs*. The *orientation* of  $\partial S$  is defined in a counter-clockwise manner, such that a point moving along the boundary would have the internal part of  $S$  on the left. The free arcs constituting  $\mathcal{L}$  inherit the orientation of  $\partial S$  and are an open subset of the topological manifold  $\partial S$ , with their endpoints on obstacles. The local obstacle boundary arcs, on the other hand, are closed in  $\partial S$ .

The *perception* of a searcher's sensor at a given pose is the tuple  $\{S, \partial S, \mathcal{L}\}$ , i.e., a footprint  $S$ , the boundary  $\partial S$ , and the set of free boundary arcs  $\mathcal{L}$  of  $\partial S$ .

## 2.2 Communication, Localization, and Memory Constraints

We require that a pair of robots are guaranteed to be able to communicate whenever their sensor footprints intersect. We further assume that two communicating robots can compute their relative poses, as a result of a mutual localization procedure (e.g., by the method described in Franchi et al (2009b), or by scan matching). We also require that a robot is able to localize itself with respect to a perception whenever it is inside the footprint. The availability of any sort of global localization is not assumed.

Each searcher must have an amount of memory which is strictly sufficient to store two perceptions, plus some variables of negligible size used for the execution of the algorithm. This constraint means that the clearing algorithm must use only a limited and constant amount of memory per robot regardless of the size of  $Q$ .

## 2.3 Inspected Region and Problem Statement

For notation and explanation, we have use for the union of the perceptions taken by all robots from different poses in  $Q$  during algorithm execution, which we refer to as the *inspected region* and denote by  $I$ . Since our algorithm does not allow recontamination,  $I$  also represents the *cleared area*. Though  $I$  will be connected, it may not be simply connected, meaning that  $\partial I$  is a set of a closed oriented curves. As with  $\partial S$ ,  $\partial I$  is oriented and partitioned into two sets: (1) the *obstacle boundary*, and (2) the *frontier* denoted by  $\mathcal{F}$ . We wish to emphasize that our algorithm does not compute or store  $I$ ,

which is incompatible with the memory constraint, but instead uses only the oriented frontier  $\mathcal{F}$ .

With these definitions we can now state the goal of our algorithm: control a team of  $n$   $(d, \phi)$ -searchers so that they always guard all the points of the frontier  $\mathcal{F}$  while expanding the cleared region  $I$  as much as possible, subject to the limited sensing, communication, and memory constraints.

## 3 The Centralized Clearing Algorithm

For clarity, we have split the presentation of our clearing algorithm into two stages. In this Section we pretend that a central controller is commanding the searchers so that we can describe the fundamental algorithm steps and the data structures involved. In Section 4 we detail the distributed implementation of the algorithm.

The team of  $n$  searchers is divided into two classes, the *frontier-guards* and the *followers*:

- *Frontier-guard*: Each frontier-guard is assigned a unique pose  $v = (x, y, \theta) \in Q \times S^1$  called the guard's *viewpoint*, which can move during the evolution of the algorithm. The frontier-guard must quickly reach its viewpoint and report a perception  $\{S, \partial S, \mathcal{L}\}$ . To detect evaders, each frontier-guard must also continuously monitor its sensor.
- *Follower*: Each follower is assigned to follow a frontier-guard, and this assignment can change as the algorithm progresses. Each follower is only required to passively follow its frontier-guard.

As needed, the central process will switch frontier-guards to followers, and vice-versa. The steps of the *centralized clearing algorithm* are as follows.

---

### Centralized Clearing Algorithm

---

Initialize one robot as frontier-guard, the rest as followers. For each  $\{S_k, \partial S_k, \mathcal{L}_k\}$  received, the central process performs the following actions:

- 1: Compute  $\mathcal{F}_k$  from  $\mathcal{F}_{k-1}$  and  $\{S_k, \partial S_k, \mathcal{L}_k\}$  as detailed in Sec. 3.1.
  - 2: Compute the next set of viewpoints  $V_{k+1}$  as detailed in Sec. 3.2.
  - 3: Assign each  $v \in V_{k+1}$  to a nearby searcher and set the searcher to be a frontier-guard.
  - 4: Assign remaining searchers a frontier-guard to follow.
  - 5: Compute paths for all frontier-guards to reach their viewpoints while maintaining coverage of  $\mathcal{F}_{k-1}$ , and send the paths to the guards.
- 

At the beginning of the algorithm, all  $n$  searchers are clustered around a point in  $Q$ . One robot is selected as the initial frontier-guard and assigned its initial pose as

**Table 1** Main symbols used in the algorithm.

<i>Symb.</i>	<i>Description</i>
$Q$	Planar environment.
$S(v)$	Sensor footprint from pose $v$ .
$S_k$	Sensor footprint of the $k$ -th perception.
$\partial S_k$	Oriented boundary of $S_k$ .
$\mathcal{L}_k$	Free boundary of $\partial S_k$ .
$I_k$	Inspected region at the $k$ -th step $:= \cup_{i=1}^k S_i$ .
$\mathcal{F}_k$	Oriented frontier arcs of $I_k$ . $\mathcal{F}_k = \mathcal{F}_{k-1}^{\text{Ext}} \cup \mathcal{L}_k^{\text{Ext}}$ .
$\mathcal{F}_{k-1}^{\text{Ext}}$	$\mathcal{F}_{k-1} \setminus \text{closure}(S_k)$ .
$\mathcal{L}_k^{\text{Ext}}$	$\mathcal{L}_k \setminus \text{interior}(I_{k-1})$ .
$V_k$	Set of viewpoints at the $k$ -th step.

a starting viewpoint. All other robots are set as followers of this guard. The frontier-guard then records the first perception, which initializes the main data stored during the evolution of the algorithm.

Whenever a frontier-guard arrives at its viewpoint and records a new perception, it sends the perception to a central processing unit. In this way the central process receives a sequence of perceptions. For each perception received, a new step  $k$  of the algorithm starts and the perception is classified as  $\{S_k, \partial S_k, \mathcal{L}_k\}$  and called the  $k$ -th perception (refer to Table 1 for a reminder of the meaning of the symbols).

We denote the total inspected region at step  $k$  as  $I_k := \cup_{i=1}^k S_i$ . Again, the algorithm does not use or store  $I_k$  or the obstacle portion of  $\partial I_k$ ; an important innovation of this work is that it stores and updates only  $\mathcal{F}_k$ , the oriented frontier arcs of  $I_k$ . Since the obstacle boundary of the inspected region  $I_k$  is impossible for either searchers or evaders to cross, there are only two ways an evader can enter  $I_k$ : (1) by being inside of  $S_k \setminus I_{k-1}$  at the instant in which the  $k$ -th perception is performed, or (2) by crossing  $\mathcal{F}_k$ . In this first case detection of the evader is immediate, the focus of our algorithm is thus on maintaining complete coverage of  $\mathcal{F}_k$  and updating it when a new perception is added.

The basic flow of the centralized clearing algorithm is as follows. The first step after receiving a new perception is to update the global frontier. The second step is to determine a new set of viewpoints to cover and expand the frontier. After that, searchers are assigned roles as guards or followers and dispatched to their respective target positions.

To guarantee the detection of any evaders in  $Q$ , the planning of new viewpoints in step 2 must meet the *frontier guarding property* and the *expansion property* laid out in the following Definition. We describe our method which achieves these properties in Section 3.2.

### Definition 1 (Viewpoint Planning Properties)

Given a non-empty frontier  $\mathcal{F}_k$  and a set of prior viewpoints  $V_k$ , the viewpoint planner selects the smallest set

of viewpoints  $V_{k+1}$  inside  $I_k$  which satisfy the following two properties:

1. Frontier guarding: Ensure  $\mathcal{F}_k$  is contained in the closure of  $\cup_{v \in V_{k+1}} S(v)$ , and
2. Expansion: Ensure  $\text{Area}(I_{k+1}) > \text{Area}(I_k)$ .

Within these constraints, the viewpoint planner maximizes  $\text{Area}(I_{k+1})$  assuming that there will be no new obstacles discovered.

We can now state the main result of this paper.

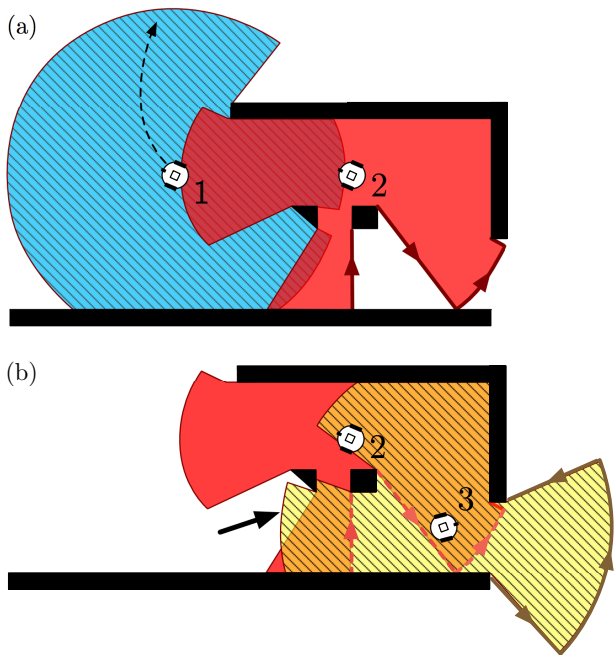
**Theorem 1 (Detection of Evaders)** *Given an implementation of the centralized clearing algorithm with the viewpoint planning properties in Definition 1, and a number of robots  $n \geq \max\{|V_k| \mid \text{for all steps } k\}$ , then the entire environment  $Q$  is cleared and every evader in  $Q$  is detected in finite time.*

*Proof.* The expansion property in Definition 1 ensures that there will be a time step  $k_f$  where  $\mathcal{F}_{k_f} = \emptyset$ , meaning that  $\partial I_{k_f}$  consists entirely of obstacle arcs and  $I_{k_f}$  completely covers  $Q$ . Therefore, for every evader  $e$  in  $Q$ , there exists at least one instant of time when  $e$  either (1) is inside of  $S_{k_e} \setminus I_{k_e-1}$  at time  $k_e \in \{1, \dots, k_f\}$ , or (2) crosses  $\mathcal{F}_{k_e-1}$  during the time interval  $[k_{e-1}, k_e]$  for  $k_e \in \{2, \dots, k_f\}$ . In the first scenario, detection of the evader is immediate. We can conclude, by means of the frontier guarding property, that the second scenario will also be detected.  $\square$

In the rest of this Section we describe how to implement the frontier update in step 1 of the algorithm and how to plan viewpoints for step 2. The path-planning in step 5 is also non-trivial, however we will only discuss how to perform this in the context of the distributed version of the clearing algorithm in Section 4.

### 3.1 Updating the Global Frontier without a Global Map

On the first iteration, frontier  $\mathcal{F}_1$  is initialized as the free boundary of the first perception,  $\mathcal{L}_1$ . For each step  $k > 1$ , the algorithm needs to compute the new frontier  $\mathcal{F}_k$ , i.e., the non-obstacle boundary of the inspected region  $I_k = I_{k-1} \cup S_k$ . The set  $\mathcal{F}_k$  can be partitioned into two subsets, (1) the set  $\mathcal{F}_{k-1}^{\text{Ext}}$  of arcs from  $\mathcal{F}_{k-1}$  which do not belong to the closure of  $S_k$ , and (2) the set  $\mathcal{L}_k^{\text{Ext}}$  of arcs from  $\mathcal{L}_k$  which are not on the interior of  $I_{k-1} = \cup_{i=1}^{k-1} S_i$ . While the computation of  $\mathcal{F}_{k-1}^{\text{Ext}}$  from  $\mathcal{F}_{k-1}$  and  $\{S_k, \partial S_k, \mathcal{L}_k\}$  is immediate, in this section we describe a novel method for computing  $\mathcal{L}_k^{\text{Ext}}$  using only the oriented arcs of  $\mathcal{F}_{k-1}$  and  $\{S_k, \partial S_k, \mathcal{L}_k\}$ .



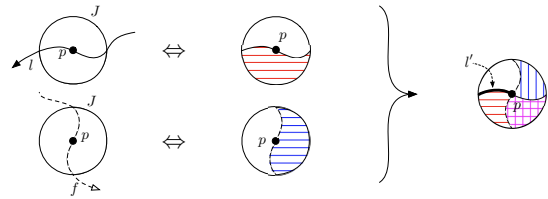
**Fig. 2** (a) After robots 1 and 2 have classified their frontiers, robot 1 moves to a new position. Once robot 1 has moved and recorded a new perception, its prior perception is no longer stored by the robot team. (b) When robot 3 arrives and records  $\{S_k, \partial S_k, \mathcal{L}_k\}$  (striped yellow), it cannot properly classify  $\mathcal{L}_k$  based only on the most recent perceptions of the other robots. Without all of  $I_{k-1}$ , robot 3 can only determine that the indicated section of  $\mathcal{L}_k$  is not on the global frontier using the intersections of  $\partial S_k$  and robot 2's oriented frontier segments (dashed red)

In all previous work including Franchi et al (2009a),  $\mathcal{L}_k^{\text{Ext}}$  has been computed using  $S_k$  and  $I_{k-1}$ . The disadvantages of this prior procedure for updating the frontier are that computing  $I_{k-1}$  requires global localization and storing it requires an amount of memory proportional to the area of environment  $Q$ , which is in contrast with the problem statement in Sec. 2.3. It is also worth noting that at step  $k$  it is not possible in general to compute  $\mathcal{L}_k^{\text{Ext}}$  using only the most recent sensor footprints from each frontier-guard, see the example in Fig. 2. The orientation of  $\mathcal{F}_{k-1}$  and  $\partial S_k$  is critically important for properly determining the frontier without  $I_{k-1}$ .

Our *global frontier update method* for computing  $\mathcal{L}_k^{\text{Ext}}$  is based around the intersections of the oriented arcs of  $\mathcal{F}_{k-1}$  and  $\partial S_k$ . Let  $\mathcal{L}_k^*$  denote the set of points belonging to the intersection between the arcs of  $\mathcal{L}_k$  and the arcs of  $\mathcal{F}_{k-1}$ , and  $\tilde{\mathcal{L}}_k^*$  the remaining points of  $\mathcal{L}_k$ . The actions of the global frontier update method are defined as follows.

#### Global Frontier Update Method

- 1: Classify the neighborhood of each  $p \in \mathcal{L}_k^*$  as internal or not
- 2: Classify the ends of each  $\ell \in \mathcal{L}_k$



**Fig. 3** Example classification of the neighborhood  $J$  of a point  $p \in \mathcal{L}_k^*$  where arcs  $\ell \in \mathcal{L}_k$  and  $f \in \mathcal{F}_{k-1}$  intersect. At left, the partitions of  $J$  induced by  $\ell$  and  $f$  are represented separately. The white region on the right side of the oriented arcs indicates the exterior, and the patterned region on the left indicates the interior. The fusion of the two partitions of  $J$  is shown at right. The bold part of  $\ell$ , denoted by  $\ell'$ , belongs to  $\mathcal{L}_k^{\text{Ext}}$  because it lies between a white and a patterned region. Note that in this case  $p \in \ell'$

- 3: Propagate classification to rest of  $\mathcal{L}_k$
- 4: Set  $\mathcal{F}_k = \mathcal{F}_{k-1}^{\text{Ext}} \cup \mathcal{L}_k^{\text{Ext}}$

The points of  $\mathcal{L}_k^{\text{Ext}}$  can be either on the boundary of or exterior to  $I_{k-1}$ , the boundary points belong to  $\mathcal{L}_k^*$  while the exterior ones belong to  $\tilde{\mathcal{L}}_k^*$ . The following crucial result states that an arc in  $\mathcal{L}_k$  can only switch between the interior and exterior of  $I_{k-1}$  at an intersection point in  $\mathcal{L}_k^*$ .

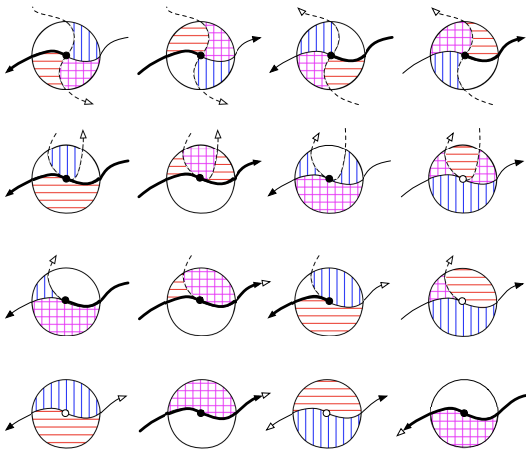
**Lemma 1 (Neighborhood Classification)** *Let  $\ell$  be an arc in  $Q$  which does not intersect  $\mathcal{F}_{k-1}$ . If any point of  $\ell$  belongs to the exterior of  $I_{k-1}$ , then all of  $\ell$  belongs to the exterior. If any point of  $\ell$  belongs to the interior of  $I_{k-1}$ , then all of  $\ell$  belongs to the interior.*

*Proof.* Since  $\ell$  is in  $Q$ , it cannot cross the obstacle boundary of  $\partial I_{k-1}$ . Therefore, if  $\ell$  does not intersect  $\mathcal{F}_{k-1}$ , then it does not cross  $\partial I_{k-1}$ .  $\square$

The first step of the frontier update method is to classify the neighborhood on  $\partial S_k$  of each intersection point  $p \in \mathcal{L}_k^*$  as either internal to  $I_{k-1}$  or not. An example of this neighborhood classification is shown in Fig. 3. The neighborhood classifications for all possible intersection cases are depicted in Fig. 4.

The second step of the method is to classify the ends of each arc  $\ell \in \mathcal{L}_k$  in the neighborhood of the endpoints of the adjacent obstacle arcs. These neighborhoods can be classified using the following Lemma.

**Lemma 2 (Obstacle Arc Classification)** *Let  $o$  denote a local obstacle arc of  $\partial S_k$ , let  $\ell_L$  and  $\ell_R \in \tilde{\mathcal{L}}_k^*$  denote the ends of the free arc segments on the left and right of  $o$ , respectively, in the neighborhood of the endpoints of  $o$ . Let  $E_o \subset o$  be the set of endpoints of any frontier arcs of  $\mathcal{F}_{k-1}$  which either begin or end on  $o$ , and which are, in the neighborhood of  $o$ , fully contained in the closure of  $S_k$ . Then:*



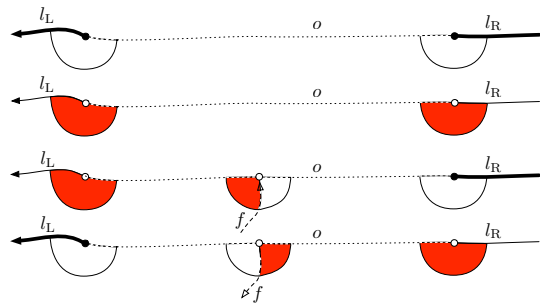
**Fig. 4** The classification of the points of arc  $\ell \in \mathcal{L}_k$  in the neighborhood of all possible types of intersections with arc  $f \in \mathcal{F}_{k-1}$ . Arc  $\ell$  is drawn solid, while  $f$  is dashed. Each row shows a different intersection type, with columns for the various reciprocal orientations of  $\ell$  and  $f$ . The first row shows isolated crossings, the second shows isolated tangents, the third shows joinings, and the fourth row shows segments where  $\ell$  and  $f$  overlap. The bold portions of  $\ell$  belong to  $\mathcal{L}_k^{\text{Ext}}$

- If  $E_o = \emptyset$ , then either  $\ell_L$  and  $\ell_R$  are both internal to  $I_{k-1}$  or neither are.
- If  $E_o \neq \emptyset$ , then  $\ell_L$  is internal to  $I_{k-1}$  if the closest<sup>1</sup>  $e \in E_o$  is the beginning of a frontier arc, and not internal otherwise. The opposite holds for  $\ell_R$ .

*Proof.* If  $E_o = \emptyset$ , as shown in the first two cases of Fig. 5, then there exists a free arc connecting  $\ell_L$  with  $\ell_R$  which is contained in the interior of  $S_k$  and is close enough to  $o$  to not intersect  $\mathcal{F}_{k-1}$ . Therefore, we can apply Lemma 1. If  $E_o \neq \emptyset$ , assume without loss of generality that it is a singleton, i.e.,  $E_o = \{e\}$ , as shown in the third and fourth cases of Fig. 5. Then, there exists a free arc connecting  $\ell_L$  to the ‘nearest’ half of the neighborhood of  $e$  which is in the interior of  $S_k$  and is close enough to  $o$  to not intersect  $\mathcal{F}_{k-1}$ . Therefore, we can apply Lemma 1. Similar claims hold for  $\ell_R$ .  $\square$

The third and final step is to propagate the classification from the neighborhoods to all points of the arcs of  $\mathcal{L}_k$ . This propagation again exploits Lemma 1. Notice that, so long as the selection of viewpoints guarantees that either  $\mathcal{L}_k^* \neq \emptyset$  or at least one local obstacle arc  $o$  has a non-empty  $E_o$ , this third step is well defined.

Combined, these three steps determine which segments  $\mathcal{L}_k$  are not in the interior of  $I_{k-1}$  and thus should be included in frontier  $\mathcal{F}_k$ .



**Fig. 5** Four classification cases are depicted for an obstacle arc  $o$  (dotted) with two adjacent free arcs (solid). In the first two, no internal frontier arc has an endpoint on  $o$ , so in the neighborhood of  $o$  the free arcs are classified as both frontier (bold) or both internal (thin). In the second two cases, an internal frontier arc  $f$  (dashed) has an endpoint on  $o$  which induces opposite classifications for the two free arcs

### 3.2 Viewpoint Planning

In this Section we describe how to pick a set of viewpoints  $V_{k+1}$  which meet the *frontier guarding property* and *expansion property* of Definition 1. With the distributed application in mind, we simplify the planning of  $V_{k+1}$  by constructing it from  $V_k$ . Let  $v_k$  be the viewpoint of the  $k$ -th perception. As detailed in Section 3.1,  $\mathcal{F}_k$  can be partitioned into two sets:  $\mathcal{F}_{k-1}^{\text{Ext}}$  (a subset of the prior frontier), and  $\mathcal{L}_k^{\text{Ext}}$  (a subset of  $\partial S_k$ ). Let  $\mathcal{F}_{k-1}^{\text{Int}}$  be the portion of  $\mathcal{F}_{k-1}$  which is inside the closure of  $S_k$ .

To construct  $V_{k+1}$ , we need the following sets:

1. The set of viewpoints  $V_k^{\text{obs}} \subset V_k$  which are assigned to guard only obsolete portions of the frontier in  $\mathcal{F}_{k-1}^{\text{Int}}$ , if any.
2. A set of new viewpoints  $V'$  inside  $S_k$  to cover and expand the new frontier segments  $\mathcal{L}_k^{\text{Ext}}$ .

With these defined, we then set:

$$V_{k+1} = ((V_k \setminus v_k) \setminus V_k^{\text{obs}}) \cup V'$$

The rest of this section is devoted to describing how to choose  $V'$  inside of  $S_k$  when  $\mathcal{L}_k^{\text{Ext}} \neq \emptyset$ .

We say that a free arc  $\ell \in \mathcal{L}_k$  is *relevant* for viewpoint planning if it contains a frontier fragment from  $\mathcal{L}_k^{\text{Ext}}$ . A relevant free arc may contain one or more frontier fragments, and each frontier fragment is entirely contained in one relevant free arc. Let  $\mathcal{L}_k^{\text{Rel}} \subseteq \mathcal{L}_k$  denote the set of relevant free arcs around  $v_k$ .

Our *local viewpoint planning method* consists of partitioning the frontier fragments of each  $\ell_{\text{Rel}} \in \mathcal{L}_k^{\text{Rel}}$  among the fewest possible new viewpoints. We first detail how to perform the method for  $d$ -searchers, that is, robots with a sensor with a field-of-view of  $2\pi$ . Afterwards, we describe how to adapt the method for  $(d, \phi)$ -searchers. In both cases, the actions of the local viewpoint planning method are as follows.

<sup>1</sup> With respect to the distance on the arc  $o$ .

---

**Local Viewpoint Planning Method**


---

Initialize  $V' = \emptyset$ . Then, for each  $\ell_{\text{Rel}} \in \mathcal{L}_k^{\text{Rel}}$  perform the following:

- 1: Determine  $p$ , the number of viewpoints needed to cover  $\ell_{\text{Rel}}$
  - 2: Partition  $\ell_{\text{Rel}}$  into  $p$  pieces
  - 3: **for**  $i = 1$  to  $p$  **do**
  - 4:   Select a pose  $v$  in  $S_k$  to cover the  $i$ -th partition of  $\ell_{\text{Rel}}$  and as much new area as possible
  - 5:   Add  $v$  to  $V'$
- 

*Remark 1* The viewpoint planner we present here is for circular sector footprints of radius  $d$  and field-of-view  $\phi \geq \pi$ . For more general footprints, our clearing algorithm could also be applied provided a viewpoint planning method with the properties in Definition 1 is available.

Each  $\ell_{\text{Rel}}$  is comprised of straight radial segments and circular segments with radius  $d$ ; see the examples in Fig. 1 and Fig. 6. The possible configurations are: single radial; single curved; curved with radial on one side; or curved with radial segments on both sides. Let  $S(v)$  denote the sensor footprint for a robot at viewpoint  $v$ . The following Lemma simplifies the determination of when a radial segment is inside  $S(v)$  for  $\phi = 2\pi$ .

**Lemma 3 (Coverage of Radial Arcs)** *Let  $v'$  be a potential new viewpoint inside  $S_k$  for a  $d$ -searcher, and let  $r \in \mathcal{L}_k^{\text{Rel}}$  be a radial free arc segment. Let  $p$  be the far endpoint of  $r$  and  $v'p$  be the line segment between  $v'$  and  $p$ . If  $\text{dist}(v', p) < d$  and  $v'p$  only intersects  $\partial S$  at  $p$ , then open set  $r$  is contained inside of  $S(v')$ .*

*Proof.* Our proof centers around the triangle  $T$  formed by  $v_k$ ,  $v'$ , and  $p$ . Radial free arc segment  $r$  is a connected subset of  $\overline{v_k p}$ . Since  $S_k$  has maximum radius  $d$ ,  $\text{dist}(v', v_k) < d$ . Combined with the fact that  $\text{dist}(v', p) < d$ , we can conclude that all of  $r$  is within  $d$  of  $v'$ . All that remains is to show that there are no obstructing obstacles inside of triangle  $T$ .

We know that  $\overline{v'p}$  is contained in the closure of  $S_k$  because it only intersects  $\partial S$  at  $p$ . Since  $S_k$  is star-shaped, both  $\overline{v_k p}$  and  $\overline{v_k v'}$  are also contained in the closure of  $S_k$ . Then, as  $S_k$  is simply connected, we can conclude that the interior of  $T$  is in  $Q$  and, therefore,  $r$  is inside of  $S(v')$ .  $\square$

There are two notable consequences of Lemma 3. First, for any  $\ell_{\text{Rel}}$  with only a radial segment, one viewpoint is sufficient. Second, for any  $\ell_{\text{Rel}}$  which contains both curved and radial segments, we only need to partition the curved segment: the viewpoint which covers

an endpoint of the curved segment will also cover any attached radial segment.

To assist in selecting viewpoints to cover curved segments, we introduce parameter  $d_{\text{min}} \in (0, d]$ , the minimum distance between  $v_k$  and any  $v \in V'$ . As will become clear,  $d_{\text{min}}$  encodes a trade-off in the algorithm: smaller values of  $d_{\text{min}}$  can reduce  $|V'|$  and thereby reduce the number of searchers required; larger values of  $d_{\text{min}}$  can increase the area exposed and thereby reduce the number of iterations required to clear  $Q$ .

Let  $\delta(\ell_{\text{Rel}})$  be the angular width of  $\ell_{\text{Rel}}$  measured counter-clockwise from the right-most frontier point on  $\ell_{\text{Rel}}$  to the left-most frontier point on  $\ell_{\text{Rel}}$ . A single viewpoint at least  $d_{\text{min}}$  from  $v_k$  can then cover an angular width of at most  $\alpha(d_{\text{min}})$  given by

$$\alpha(d_{\text{min}}) = 2 \arccos(d_{\text{min}}/2d) \in \left[\frac{2\pi}{3}, \pi\right).$$

The number of viewpoints  $\eta$  necessary to cover  $\ell_{\text{Rel}}$  is then determined by the following Lemma.

**Lemma 4 (Number of Viewpoints Required)** *For any  $\ell_{\text{Rel}} \in \mathcal{L}_k^{\text{Rel}}$ , the clearing algorithm requires  $\eta \in \{1, 2, 3\}$  viewpoints. Moreover:*

- if  $\delta(\ell_{\text{Rel}}) \leq \frac{2\pi}{3}$ , then  $\eta = 1$ ,
- if  $\frac{2\pi}{3} < \delta(\ell_{\text{Rel}}) < \pi$ , then  $\eta = 1$  or  $2$ ,
- if  $\pi \leq \delta(\ell_{\text{Rel}}) < 2\pi$ , then  $\eta = 2$  or  $3$ , and
- if  $\delta(\ell_{\text{Rel}}) = 2\pi$ , then  $\eta = 3$ .

*Proof.* This result is a direct consequence of Lemma 3 and the fact that  $\alpha(d_{\text{min}}) \in \left[\frac{2\pi}{3}, \pi\right)$ .  $\square$

For  $\eta > 1$ , the angular width of  $\ell_{\text{Rel}}$  is then partitioned such that the first viewpoint covers  $[0, \delta(\ell_{\text{Rel}})/\eta]$ , and each subsequent viewpoint covers the next equally sized slice of angular width. This partitioning of  $\ell_{\text{Rel}}$  achieves step 2 of the viewpoint planning method.

After partitioning  $\ell_{\text{Rel}}$ , the final step is to place each new viewpoint  $v$ . This placement must ensure that a perception from  $v$  covers the required portion of  $\ell_{\text{Rel}}$  and also uncovers as much area as possible beyond  $\ell_{\text{Rel}}$  (assuming no new obstacles). For single radial segments, we place  $v$  at the midpoint of the segment facing perpendicular to the segment out into the unknown territory beyond  $\ell_{\text{Rel}}$ . For all other configurations, we construct a line through  $v_k$  which bisects the curved arc in  $\ell_{\text{Rel}}$  assigned to  $v$ . We then place  $v$  on this bisector at the point in  $S_k$  which is closest to the intersection with  $\ell_{\text{Rel}}$  and also ensures that both endpoints of the curved arc in  $\ell_{\text{Rel}}$  assigned to  $v$  will be inside  $S(v)$ . Here pose  $v$  is oriented radially outward from  $v_k$ .

By construction, this method of selecting  $V'$  guarantees that  $\mathcal{L}_k^{\text{Ext}} \in \cup_{v \in V'} S(v)$  for searchers with  $\phi = 2\pi$ , meaning that the *frontier guarding property* in Definition 1 is satisfied. For  $d_{\text{min}}$  close to zero, it creates

the fewest new viewpoints possible, while for  $d_{\min} = d$  it exposes more area with minimal additional viewpoints. The following Lemma shows that this viewpoint planner also guarantees the *expansion property*, as  $\text{Area}(I_{k+1})$  will be greater than  $\text{Area}(I_k)$ .

**Lemma 5 (Guaranteed Expansion)** *For each  $v \in V'$ ,  $S(v)$  includes some new area  $A \in S(v)$  where  $\text{Area}(A) > 0$  and  $\text{Area}(A \cap I_k) = 0$ .*

*Proof.* Let  $f \in \ell_{\text{Rel}}$  be a frontier segment assigned to  $v$ . By definition,  $f$  is inside of  $Q$  and is a subset of  $\mathcal{F}_k$ . By Lemma 3,  $f$  is also inside of the open set  $S(v)$ , so there must be an open set  $A \subset Q$  which is inside of  $S(v)$  but outside of  $I_k$ .  $\square$

We have described a viewpoint planning method which meets the requirements of Definition 1 for searchers with  $\phi = 2\pi$ . For  $(d, \phi)$ -searchers whose sensors have a field-of-view in  $[\pi, 2\pi)$ , the above method is optimal when  $\ell_{\text{Rel}}$  contains either only radial frontier fragments, or only curved frontier fragments. One option for handling mixed fragments is to split them and handle the radial and curved parts separately. However, this simple approach may create more viewpoints than strictly required. We instead propose the following geometric method.

Consider the case when  $\ell_{\text{Rel}}$  consists of a radial segment on the right of a curved segment. Let  $p_r$  be the first frontier point in the radial part of  $\ell_{\text{Rel}}$ , and let  $p_m$  be the intersection of the curved and radial segments. Next, loop over possible  $p_l$ 's, starting from  $p_l = p_m$  and moving along the curved segment, stopping at the furthest  $p_l$  for which the midpoint of  $\overline{p_r p_l}$  is within  $d$  of  $p_m$ . Then, place  $v$  at the midpoint of  $\overline{p_r p_l}$ , facing outward perpendicular to  $\overline{p_r p_l}$ . This placement ensures that all frontier points between  $p_r$  and  $p_l$  are covered by a perception taken from  $v$  for  $\phi \geq \pi$ , while maximizing the amount of  $\ell_{\text{Rel}}$  covered. If any frontier on the curved segment remains uncovered, it can be handled using the prior approach. This method can be trivially modified if the radial segment is on the left, and can also be applied on both sides for a curved segment with radial segments on both sides.

#### 4 The Distributed Clearing Algorithm

In the distributed setting, the communication graph is in general disconnected, necessitating some changes from the centralized description. First, the global frontier must be stored and updated in a distributed manner. Second, viewpoint planning must be performed locally by the frontier-guards. Furthermore, the distributed algorithm only has access to pairwise relative

mutual localization between neighbors. Finally, while the centralized version is synchronous and sequential, the distributed setting is asynchronous and concurrent, i.e., it is possible for perceptions from disconnected searchers to be recorded at the same time.

#### 4.1 Distributed Handling of Global Frontier and Viewpoint Planning

We distribute the storing and updating of the global frontier by having each frontier-guard store its local frontier segments and update them through communication with its neighboring frontier-guards. We denote the section of the global frontier  $\mathcal{F}_k$  owned by robot  $i$  by  $\mathcal{F}_{k,i}$ . This distributed storage of the global frontier can always be achieved since, by the frontier guarding property, each global frontier point is guarded by a frontier-guard.

The *pairwise frontier update method* which follows is a distributed version of the method in Sec. 3.1 for classifying the free boundary  $\mathcal{L}_k$ .

---

#### Pairwise Frontier Update Method

---

When robot  $i$  records a new perception, it updates  $\mathcal{F}_k$  as follows:

- 1: Classify neighborhood of each intersection  $p$  between  $\mathcal{L}_k$  and  $\mathcal{F}_{k-1,i}$  as internal or not, if any
  - 2: **for** each robot  $j$  in communication with  $i$  **do**
  - 3:   Classify neighborhood of each intersection  $p$  between  $\mathcal{L}_k$  and  $\mathcal{F}_{k-1,j}$  as internal or not
  - 4:   Inform  $j$  if any piece of  $\mathcal{F}_{k-1,j}$  lies inside  $S_k$
  - 5:   Classify the ends of each  $\ell \in \mathcal{L}_k$
  - 6:   Propagate classification to rest of  $\mathcal{L}_k$
  - 7:   Store  $\mathcal{F}_{k,i}$
- 

This distributed frontier classification is always possible because the classification of  $\mathcal{L}_k$  requires only the frontier fragments from  $\mathcal{F}_{k-1}$  which intersect  $S_k$ . In the distributed setting, each of these frontier fragments belong either to a neighboring guard's perception or to robot  $i$ 's previous perception. The localization with respect to the first kind of fragments is guaranteed since by assumption two robots whose footprints intersect are in communication and are mutually localized. The localization w.r.t. the second kind of fragments is also guaranteed by assumption since robot  $i$ 's current viewpoint lies in the footprint of the previous one.

Using the pairwise frontier update method, updates to the global frontier are based only on current relative poses of nearby searchers, not on absolute poses. The distributed clearing algorithm, therefore, is able to continue clearing an environment even if the searchers cannot determine where they are relative to where they started. Note that, because it operates in pairs,



this frontier update method requires only an amount of memory per robot proportional to that required to store two perceptions.

Once  $\mathcal{F}_{k,i}$  is determined, we can use the local viewpoint planning method from Sec. 3.2. This method is already distributed as it requires only the local frontier of the frontier-guard doing the planning. The execution of the path to new viewpoints can also be done without global localization, whether it's being executed by the guard itself or by a follower. Since the new viewpoint lies inside the local perception, either local odometry of reasonable accuracy or a registration of footprints taken along the path with  $S_k$  will suffice.

#### 4.2 Distributed Algorithm & Robot Roles

The two classes of searchers from the centralized algorithm are each split in two, yielding four possible states:

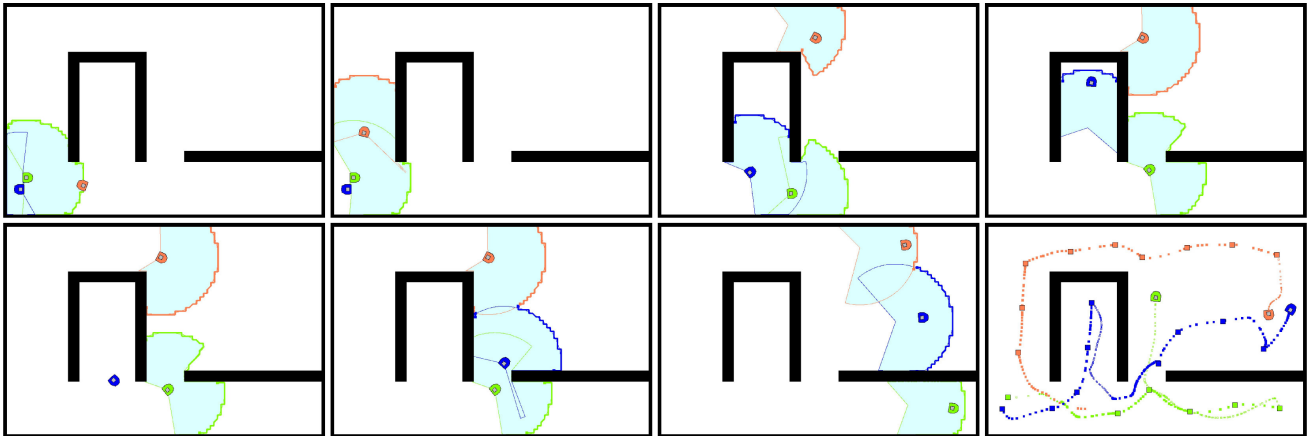
- *Expand*: When a searcher is assigned a new viewpoint to move to, it enters the expand state until it reaches the viewpoint and records a perception.
- *Frontier-guard*: Each frontier-guard  $i$  remains stationary at its viewpoint and has complete control over its local frontier segments,  $\mathcal{F}_{k,i}$ . It must communicate with neighboring frontier-guards to update  $\mathcal{F}_{k,i}$ , plan a new viewpoint to cover and expand  $\mathcal{F}_{k,i}$ , and dispatch a follower to the new viewpoint.
- *Follow*: Must passively follow and respond to commands from a frontier-guard or expander.
- *Wander*: When a frontier-guard no longer has a local frontier to guard, it wanders until it locates a leader to follow.

The *distributed clearing algorithm* detailed in Table 2 consists of an initialization step, followed by each searcher iteratively executing the procedure corresponding to its current state. These procedures have subroutines for all important computations, and detail when searchers transition between states. There are four key subroutines we would like to highlight:

- *UpdateNeighbFrontier/Frontier*: These two functions perform the pairwise frontier update method in Section 4.1.
- *ViewPointPlan*: This function follows the local viewpoint planning method in Section 3.2, and then picks the best new viewpoint to expand first.
- *PathToViewPoint*: Determines a safe path from the current viewpoint to the new viewpoint inside  $S$ , which can be a straight line since  $S$  is star-shaped.
- *SearchForLeader*: This function does a random walk with two additional behaviors: 1) when it encounters a frontier-guard, it switches to Follow; 2) wanderers may join to form a wandering blob.

**Table 2** Details of the distributed clearing algorithm

<b>Distributed Clearing Algorithm</b>	
To begin, one searcher is set to Expand to its starting pose and all others start either Following the first or in the Wander state. All agent's then continuously execute the procedure corresponding to their state:	
<b>Procedure Expand</b>	
<b>Data:</b> frontier,path <b>1</b> <b>foreach</b> follower <i>in</i> followers <b>do</b> <b>2</b>   Send(follower, "follow",path); <b>3</b> Move(path); <b>4</b> { $S, \partial S, \mathcal{L}$ } $\leftarrow$ Perceive(); <b>5</b> neighbFront $\leftarrow$ UpdateNeighbFront(); <b>6</b> frontier $\leftarrow$ Frontier({ $S, \partial S, \mathcal{L}$ },frontier,neighbFront); <b>7</b> DoBehavior("Frontier-Guard", $S$ ,frontier);	
<b>Procedure Frontier-Guard</b>	
<b>Data:</b> $S$ ,frontier <b>1</b> <b>if</b> frontier <i>is empty</i> <b>then</b> <b>2</b>   Send(followers, "wander"); <b>3</b>   DoBehavior("Wander"); <b>4</b> (bestVP,NumVPs) $\leftarrow$ ViewPointPlan( $S$ ,frontier); <b>5</b> path $\leftarrow$ PathToViewPoint( $S$ ,bestVP); <b>6</b> <b>if</b> NumVPs == 1 <b>then</b> <b>7</b>   DoBehavior("Expand",frontier,path); <b>8</b> <b>else</b> <b>9</b>   <b>if</b> followers <i>has at least one follower</i> <b>then</b> <b>10</b>       follower $\leftarrow$ PopFollower(followers); <b>11</b>       Send(follower, "expand",path); <b>12</b>       WaitForFollower(follower); <b>13</b>   <b>else</b> <b>14</b>       <b>while</b> <i>no new neighbor and no followers</i> <b>do</b> <b>15</b>           Sleep(); <b>16</b>   DoBehavior("Frontier-Guard", $S$ ,frontier);	
<b>Procedure Follow</b>	
<b>1</b> Receive(Leader,message,path); <b>2</b> <b>switch</b> message <b>do</b> <b>3</b>   <b>case</b> "follow" <b>4</b>       Move(path); <b>5</b>   <b>case</b> "expand" <b>6</b>       DoBehavior("Expand", $\emptyset$ ,path); <b>7</b>   <b>case</b> "wander" <b>8</b>       DoBehavior("Wander");	
<b>Procedure Wander</b>	
<b>1</b> SearchForLeader(); <b>2</b> <b>if</b> leader <i>found</i> <b>then</b> <b>3</b>   DoBehavior("Follow"); <b>4</b> <b>if</b> <i>all searchers wandering</i> <b>then</b> <b>5</b>   <b>exit</b>	



**Fig. 6** Simulation of three  $(d, \phi)$ -searchers clearing an environment. Recorded perceptions are shown in a light blue, with frontiers shown with bold lines in the color of the frontier guard who owns them. The trajectories of the robots are shown in the final panel, with large squares for viewpoints

The following subsections describe our implementation of the algorithm, show simulation and hardware experiment results, and expand on some technical details.

#### 4.3 Illustrative Simulation

To demonstrate the utility of the proposed distributed clearing algorithm, we implemented it using the open-source Multirobot Integration Platform (MIP) (Franchi and Stegagno, 2010) and the Player/Stage robot software system (Gerkey and contributors, 2009). The clearing algorithm and related modules were implemented using the MIP architecture, which provides a multi-tasking estimation/control framework, a realistic simulation environment, and allows direct porting for execution on real robots. Perceptions are implemented as local coverage grids with  $5\text{cm}$  resolution, with oriented frontier arcs handled as ordered sequences of cells. Each robot stores only its most recent perception and its local frontier.

Figure 6 provides a detailed example of three robots implementing the distributed algorithm. A video is also available in Online Resource 1. The searchers are simulated Khepera III robots with laser rangefinders with a range of  $0.8\text{ m}$  and a field-of-view of  $240^\circ$ . Perfect mutual localization is provided for this simulation, while the Smooth Nearness Diagram navigation driver in Player is used to navigate between viewpoints and avoid collisions (Durham and Bullo, 2008).

The first panel of Fig. 6 shows the initialization of the algorithm. The three robots start within communication range of each other and with initial poses which do not significantly interfere with each other’s sensors. The green robot begins as a frontier-guard and records

the first perception. The blue robot then clears the area behind the green robot.

In the second panel, the orange robot has expanded one of the initial frontiers, classified its boundary, and become a frontier-guard. As orange only needs one new viewpoint to expand its single frontier arc, it will expand alone around the top of the obstacle. Blue is then dispatched to clear the other initial frontier.

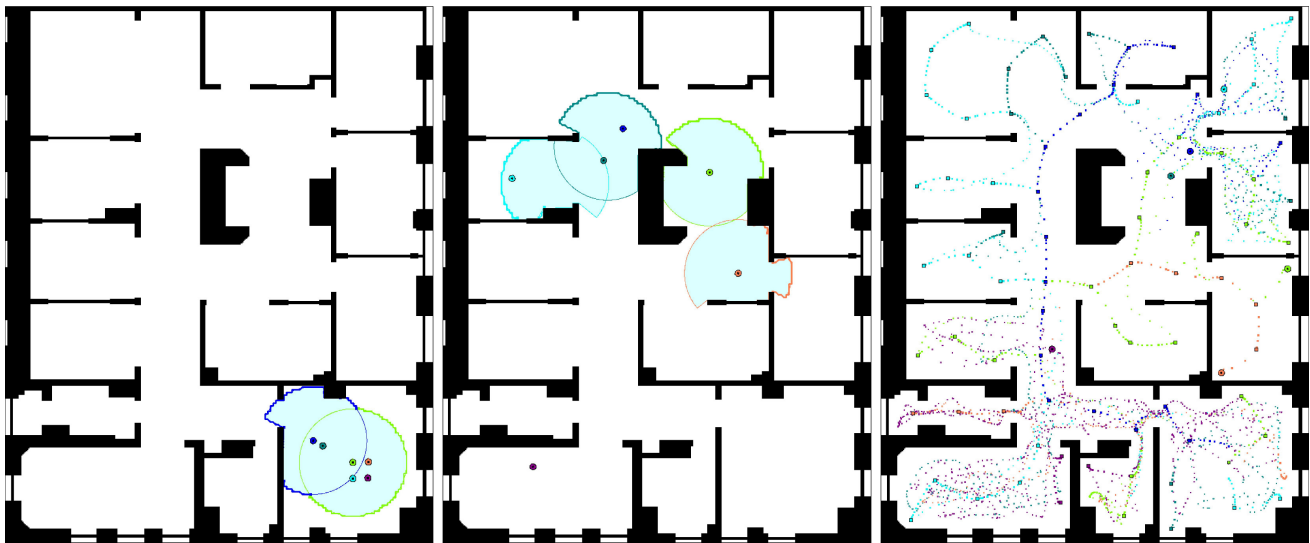
The next two images show the continued expansion of the cleared area. By the fourth panel, both orange and green have reached positions from which they require assistance in order to expand. After the blue robot clears the inside of the U-shaped obstacle, it enters the Wander state and searches for a leader.

The remaining images show the final stages of the algorithm, where orange and blue clear one room while green clears the lower corridor. Green finishes before the others, and enters the Wander state to try to find them. The final panel shows recorded trajectories for the robots during clearing as well as all viewpoints.

#### 4.4 Hospital Wing Simulation

Figure 7 presents a larger simulation in a more complex environment modeled after part of a wing of a hospital. A movie version is also available in Online Resource 2. The environment is  $16\text{m}$  wide and  $20\text{m}$  tall, with a number of small patient rooms around a central desk, as well as a couple storage closets and other rooms at the bottom of the map.

Six simulated  $d$ -searchers with  $d = 2.0\text{m}$  begin in the largest room in the bottom right corner. The first image shows the result of the first expansion, with the blue robot having reached its viewpoint and erasing the first third of the initial full circle frontier. The initial



**Fig. 7** Three screenshots from a simulation of six  $d$ -searchers clearing a portion of a hospital wing. The paths of the agents are shown at right, with all viewpoints drawn with larger squares

frontier-guard then dispatches another follower to cover the second third, before expanding the final third itself.

By the middle image, the team has swept through all of the lower rooms. Five of the searchers are engaged in covering and expanding the frontier, while the purple robot remains behind. The purple robot was part of the group of four which cleared the bottom left room, and all four of those searchers entered the Wander state once that room was clear. While most of these searchers found their way to an active frontier, the purple searcher is still wandering.

The final image shows the recorded trajectories of each agent, with viewpoints indicated with larger squares. The bottom set of rooms, as well as the rooms in the top right, show a dense set of tracks of searchers. The density in these rooms is a result of multiple searchers repeatedly executing the Wander behavior after clearing these parts of the map. The clear lines in the top right and middle left show the efficient, simple paths taken by searchers executing repeated expansions.

#### 4.5 Hardware Experiments

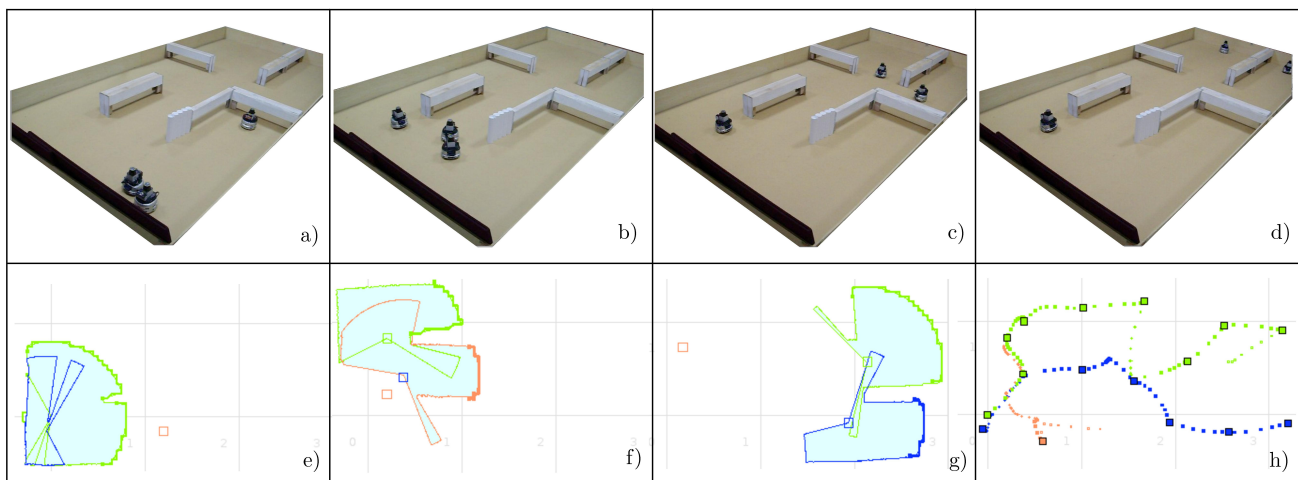
The distributed clearing algorithm was experimentally validated using Khepera III robots. Each robot is equipped with a wi-fi card and a Hukuyo URG-04LX laser sensor. The latter has a field-of-view of  $240^\circ$  and a range artificially limited to  $0.8\text{ m}$ . Simple odometry is used to provide mutual localization and Smooth Nearness Diagram navigation is used to avoid obstacles. Each robot is controlled by a separate process and they communicate with each other using a wireless network.

A complete experiment is summarized in Fig. 8, where each column contains a camera image and the relative perceptions for a distinct phase of the algorithm. A video of the experiment is provided in Online Resource 3. In the first two panels (a,e) one robot acts as frontier-guard while the others are followers. By the second phase (b,f), the first dead-end corridor has been cleared and two frontier-guards are set to sweep the next two corridors. In the third phase (c,g) one robot simulates a motor fault, which forces the two other robots to complete the task by themselves. In the end (d,h), the environment is fully cleared and the trajectories for each robot are shown, with larger boxes indicating viewpoints.

## 5 Theoretical Analysis & Results

### 5.1 Frontier Guarding & Expansion Properties

The behavior of the frontier-guards in the distributed clearing algorithm guarantees both the frontier guarding property and the expansion property from Definition 1. When expander  $i$  reaches its viewpoint and makes a perception, it then enters the stationary frontier-guard state. So long as  $i$  remains a frontier-guard, it maintains complete coverage of the frontier segments in  $\mathcal{F}_{k,i}$ . Searcher  $i$  will only leave the frontier-guard state if either  $\mathcal{F}_{k,i}$  is erased by a new neighbor, or if  $i$  determines that one new viewpoint is sufficient to cover  $\mathcal{F}_{k,i}$  and that the path to the viewpoint also maintains coverage of  $\mathcal{F}_{k,i}$ . The local viewpoint planning method guarantees that each new viewpoint will expand the cleared area.



**Fig. 8** Four phases of an experiment with three Khepera III robots with Hukuyo URG-04LX laser sensors. One of the robots simulates a motor fault (b,f) which forces the others to complete the task by themselves (d,h)

## 5.2 Algorithm Completeness

Two assumptions are required to extend Theorem 1 and claim that the distributed clearing algorithm is guaranteed to detect every evader in  $Q$ . First, there must be a sufficient number of searchers available to expand at least one frontier segment at each step of the algorithm. Second, any searcher who enters the Wander state must reach an active frontier in finite time. When these two assumptions are satisfied, the searchers will never have to wait an infinitely long time between the recording of perceptions. Therefore, we can conclude following the proof of Theorem 1 and the frontier guarding and expansion properties that the distributed clearing algorithm will clear all of  $Q$  and detect every evader in  $Q$ .

## 5.3 Bounds on Number of $d$ -Searchers Required to Clear Tree-like Environments

A well-known result from the literature is that computing the minimum number of searchers required to clear a general graph is NP-hard (Parsons, 1978). This result was extended in Guibas et al (1999) to searchers with infinite range sensors in polygonal environments, and so solving for the minimum number of  $d$  or  $(d, \phi)$ -searchers to clear a non-polygonal environment is also NP-hard. Instead, we examine a simpler illustrative scenario.

Consider an environment made of narrow hallways which intersect at junctions. Each hallway section between two junctions is narrower than  $2d$  and longer than  $2d$ , meaning that having a single  $d$ -searcher traverse the hallway is both sufficient and necessary to clear the hallway. In addition, assume that one robot placed in a junction can detect any evader which enters the junc-

tion. With these assumptions, we can model the environment as a graph where hallways are represented by edges of unitary length, junctions by nodes with degree three or greater, and any dead-end hallways as nodes of degree one.

The following two Propositions give bounds on the number of agents required for the special case where the graph representing the environment is a tree. Let us assume, without loss of generality, that the robots start at a junction. We consider two strategies for dispatching followers: (1) assign all available followers to follow an expander (depth-first strategy), and (2) a general follower allocation strategy.

**Proposition 1 (Depth-first on Tree)** *Let  $h$  be the maximum number of hops from the starting node to a leaf of the tree. The number of robots required to clear the environment with the distributed clearing algorithm using a depth-first strategy is upper bounded by: (i)  $h$  if the starting node is a leaf, or (ii)  $h + 1$  otherwise.*

*Proof.* From the starting node, the team will initially expand through one path in the tree, with one robot staying as a guard in each non-leaf node in the path. The longest such path will require  $h - 1$  guards if the starting node is a leaf, and  $h$  guards otherwise. One additional robot is needed to expand and clear the leaf node at the end of the path. Hence the result follows.  $\square$

**Proposition 2 (General Tree Bound)** *Let  $\ell$  be the number of nodes in the tree which have a leaf node as a child. Then, the number of robots required to clear the environment with the distributed clearing algorithm using any follower allocation strategy is upper bounded by  $\ell + 1$ .*

*Proof.* Expanding out from the starting node, the maximum number nodes which could require guarding at the same time is equal to  $\ell$ . If one additional agent was available, this agent would be dispatched to expand and clear a leaf node. Once the leaf node was clear, the agent would wander back to the parent node and then continue on to the next leaf, eventually clearing all the leaves and thus the full tree.  $\square$

#### 5.4 Time and Space Computational Complexity

The four significant subroutines in our algorithm have minimal computational requirements. One important innovation of this work is the pairwise frontier merging method, which requires only  $\mathcal{O}(2|\partial S|)$  memory and  $\mathcal{O}(|\partial S|^2)$  time to find intersections and classify the local frontier. Our geometric viewpoint planning method typically requires only constant time per viewpoint, but scales with  $|\partial S|$  if  $\phi < 2\pi$  and the relevant frontier arc consists of both curved and radial segments. Path planning to new viewpoints is trivial as the straight line between viewpoints is always in  $S$ . Reactive collision avoidance can then be handled by a local planner like Smooth Nearness Diagram navigation. Finally, the search for leader subroutine is also straightforward as it must simply pick a point on the free boundary of the robot's current sensor footprint to drive towards.

#### 5.5 Detecting Completion

When the environment has been cleared, all searchers will be in the Wander state. If all-to-one communication is available (e.g., if all robots have even a very low-bandwidth connection to a central command center), then detecting task completion is trivial. In the most general case, the wandering searchers will have to reach a consensus that the task is complete by querying other searchers when they encounter them. In the absence of global localization or other means of assuring rendezvous, our proposal is that robots in the Wander state clump together when they encounter each other to form wandering blobs. Eventually, through the random walks of these growing blobs, all searchers will be joined into a single blob and task completion can be easily detected.

#### 5.6 Handling Agent Failure

The distributed clearing algorithm relies on maintaining complete coverage of the global frontier at all times. The failure of any searcher in the frontier-guard

state, therefore, has the potential to recontaminate the cleared area and require restarting the algorithm. However, the algorithm can be made quite robust to random failures with a few minor modifications, at the cost of requiring a larger robot team.

The two mission-critical robot behaviors are Frontier-guard and Expand. To handle the potential failure of a frontier-guard, all followers of the guard could hold duplicate copies of the guard's perception and local frontier. The followers would regularly communicate with the guard to check that it is functioning and, if it fails, then one follower would take its place. If a high degree of robustness is required for a particular application, the algorithm could be modified to ensure each guard always has one or more followers. The failure of a searcher in the Expand state could be handled by one of its followers in a similar manner. In addition, when a frontier-guard commands a searcher to expand to a particular point, it can regularly check the expander's progress and dispatch another agent if necessary.

## 6 Numerical Analysis & Results

In this section we use a Monte Carlo probability estimation method from Tempo et al (2005) to place some probabilistic bounds on how much area 12  $d$ -searchers can clear in an infinite obstacle-free environment. The simulated searchers expand from their starting position and clear as much area as they can before reaching a final equilibrium where the team would need additional searchers to continue. The theoretical limit on the area cleared by 12  $d$ -searchers occurs when the searchers are at the vertices of a 12-sided regular polygon with sides of length  $2d$ . The cleared area in this limit is the area of the regular polygon plus the area of the sensor footprint of each searcher beyond the polygon. For  $n$  searchers, this area is given by

$$\text{Cleared area limit} = \frac{nd^2}{\tan\left(\frac{\pi}{n}\right)} + \frac{(n+2)\pi d^2}{2}.$$

We set  $d = 1.0$ , meaning the maximum clearable area for 12 searchers is  $66.78m^2$ .

The Chernoff bound describes the minimum number of random samples  $N$  required to reach a certain level of accuracy in a probability estimate. For an accuracy  $\epsilon \in (0, 1)$  and confidence  $1 - \eta \in (0, 1)$ , the number of samples is given by

$$N \geq \frac{1}{2\epsilon^2} \log \frac{2}{\eta}.$$

For  $\eta = 0.01$  and  $\epsilon = 0.1$ , at least 116 samples are required.

In each trial, we chose a random agent as the initial frontier-guard, which produced differences in subsequent robot roles and timing of establishment of perceptions. In addition, the robots are only asked to get within 2 *cm* and 2° of a particular viewpoint, which leads to variability in the resulting perceptions and frontiers.

Over 118 simulations, the 12 searchers cleared an average of 93.4% of the optimum area with a standard deviation of 2.3%. The maximum percentage cleared was 97.3%, while the minimum was 88.5%. A video of an example trial which cleared 96.2% of the optimal area is available as Online Resource 4. Variability in the area cleared came from differences in the overlap between searcher footprints and how closely the final shape resembled a circle.

Based on these results, we can conclude that with 99% confidence there is at least a 90% probability that 12 *d*-searchers executing the distributed clearing algorithm in a large empty environment will clear at least 88.5% of the maximum possible area.

## 7 Conclusion & Future Work

We have presented a distributed pursuit-evasion algorithm for a team of mobile robots with limited sensing and communication capabilities, limited on-board memory, and access to only local mutual localization. Our algorithm can guarantee detection of moving evaders in an unknown, non-polygonal environment with holes, provided the team consists of a sufficient number of robots. A key contribution of this algorithm is a novel method for updating the global frontier between cleared and contaminated regions using only local information. We also validated the algorithm through both simulations and hardware experiments, and discussed some theoretical and numerical results on the algorithm's performance.

There are a number of interesting future directions for this work. One useful extension would be to guarantee a connected communication graph for the searchers at all times, perhaps including a connection back to the initial starting point. The development of bounds on the number of *d*-searchers required to clear a general environment would be a significant contribution. Finally, the frontier concept could also be applied to three-dimensional environments.

**Acknowledgements** The authors would like to thank Paolo Stegagno and the Robotics Lab at DIS led by Giuseppe Oriolo. This work is supported in part by National Science Foundation award IIS-0904501 and by Army Research Office MURI award W911NF-05-1-0219.

## References

- Adler M, Racke H, Sivadasan N, Sohler C, Vocking B (2003) Randomized pursuit-evasion in graphs. *Combinatorics, Probability and Computing* 12(3):225–244
- Bopardikar SD, Bullo F, Hespanha JP (2008) On discrete-time pursuit-evasion games with sensing limitations. *IEEE Trans on Robotics* 24(6):1429–1439
- Durham JW, Bullo F (2008) Smooth nearness-diagram navigation. In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Nice, France, pp 690–695
- Franchi A, Stegagno P (2010) Multirobot Integrated Platform. <http://www.dis.uniroma1.it/~labrob/software/MIP/>
- Franchi A, Freda L, Oriolo G, Vendittelli M (2009a) The sensor-based random graph method for cooperative robot exploration. *IEEE/ASME Trans on Mechatronics* 14(2):163–175
- Franchi A, Oriolo G, Stegagno P (2009b) Mutual localization in a multi-robot system with anonymous relative position measures. In: *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, St. Louis, MO, pp 3974–3980
- Gerkey B, contributors (2009) The Player/Stage Project. <http://playerstage.sourceforge.net>, version 2.12
- Gerkey BP, Thrun S, Gordon G (2006) Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research* 25(4):299–315
- Guibas LJ, Latombe JC, Lavalley SM, Lin D, Motwani R (1999) A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry & Applications* 9(4/5):471–494
- Hollinger G, Singh S, Kehagias A (2010) Improving efficiency of clearing with multi-agent teams. *International Journal of Robotics Research* 29(8):1088–1105
- Howard A, Mataric MJ, Sukhatme GS (2002) An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots* 13(2):113–126
- Jung B, Sukhatme GS (2002) Tracking targets using multiple robots: The effect of environment occlusion. *Autonomous Robots* 13(3):191–205
- Kolling A, Carpin S (2008) Multi-robot surveillance: an improved algorithm for the graph-clear problem. In: *2008 IEEE Int. Conf. on Robotics and Automation*, Pasadena, CA, pp 2360–2365
- Kolling A, Carpin S (2010) Multi-robot pursuit-evasion without maps. In: *2010 IEEE Int. Conf. on Robotics and Automation*, Anchorage, Alaska, pp 3045–3051
- Parsons TD (1978) Pursuit-evasion in a graph. In: Alavi Y, Lick D (eds) *Theory and Applications of Graphs*, Lecture Notes in Mathematics, vol 642, Springer, pp

426–441

- Sachs S, Rajko S, LaValle SM (2004) Visibility-based pursuit-evasion in an unknown planar environment. *International Journal of Robotics Research* 23(1):3–26
- Suzuki I, Yamashita M (1992) Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing* 21(2):863–888
- Tempo R, Calafiore G, Dabbene F (2005) *Randomized Algorithms for Analysis and Control of Uncertain Systems*. Springer
- Yamauchi B (1998) Frontier-based exploration using multiple robots. In: 2nd Int. Conf. on Autonomous Agents, Minneapolis, MN, pp 47–53