Distributed Pursuit-Evasion without Mapping or Global Localization via Local Frontiers

Joseph W. Durham · Antonio Franchi · Francesco Bullo

Received: 29 October 2010 / Accepted: 5 November 2011

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 to guarantee evader detection 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 map of the environment or requiring global localization. We demonstrate the functionality of the algorithm through simulations in realistic environments and through hardware experiments. We also compare Monte Carlo results for our algorithm to the theoretical optimum area cleared as a function of the number of robots available.

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.

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

Corresponding author: 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

1.1 Related Work

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 the case of 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, 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 is inspired by methods for exploration and deployment based on the frontier between explored and unknown regions (Yamauchi, 1998; Howard et al, 2002; Franchi et al, 2009a). A preliminary version of this work appeared in (Durham et al, 2010).

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, ϕ)-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.

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 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 discetized 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 as a function of the number of robots.

In the literature, Kolling and Carpin (2010) is the only other work for multiple robots which can guarantee detection of evaders without prior knowledge of the environment. Their approach sweeps hallways using lines of robots where the robots at the ends perform wall following and give commands to those in the middle. There are a few important differences from our work. For one, they build and store a topological map whose memory footprint scales with the size of the environment, whereas our approach requires only a constant amount of memory per robot. Partly because of the global map, their approach sweeps only one hallway at a time and they admit there are complications in dealing with topological holes. Our method expands in parallel and handles holes seemlessly. Their dependence on wall following would also prove challenging in cluttered environments or in large empty spaces which our approach can handle. We also provide simulation and experimental validation which are not included in their theoretically-oriented work.

1.3 Paper Organization

Section 2 provides definitions and states the problem we are addressing. In Section 3 we examine a centralized version of our algorithm to explain some details. The decentralized clearing algorithm is presented and illustrated in Section 4. In Section 5 we discuss theoretical properties of the algorithm and in Section 6 we present experimental results. We conclude in Section 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 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 d-searchers, which are a (d, ϕ)-searchers 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

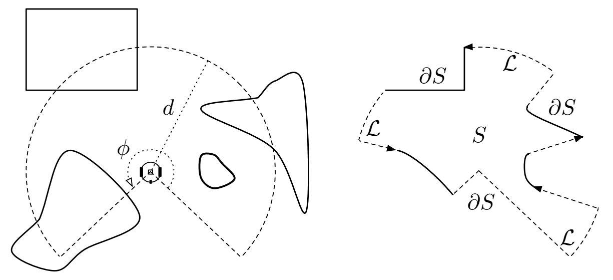


Fig. 1 On the left, four obstacles surround a $(d, \phi)-$searcher and lie within the dashed circular sector representing the area perceivable by the searcher's sensor without occlusions. The right image shows the boundary ∂S of the sensor footprint for this configuration, with dashed oriented arcs for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

boundary of the sensor footprint, ∂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 ∂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 ∂S and are an open subset of the topological manifold ∂S, with their endpoints on obstacles. The local obstacle boundary arcs, on the other hand, are closed in ∂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 ∂S, and the set of free boundary arcs $\mathcal{L}$ of ∂S.

2.2 Communication, Localization, and Memory

Our method for classifying ∂S requires that a pair of robots are guaranteed to be able to communicate whenever their sensor footprints intersect. For example, this condition would be satisfied if robots can communicate when the distance between them is less than the sum of the radii of their sensor footprints.

One of the benefits of our approach is that it can work in the absence of global localization. Instead, we will assume that two robots with intersecting sensor footprints can compute their relative poses as a result of some mutual localization procedure. Mutual localization could be achieved by the method described in Franchi et al (2009b), or by scan matching (Censi, 2008). Alternatively, the mutual visibility of the overlapping portion of footprints could be used by projecting calibration dots or dispatching an extra robot to serve as an intermediary. We further require that a robot is able to localize itself with respect to a perception whenever it is inside the footprint, e.g., by a simple pairwise scan matching.

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 each step of the distributed 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 ∂I is a set of closed oriented curves. As with ∂S, ∂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, ϕ)-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.

At any given time, 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 $v = (x, y, \theta) \in Q \times [0, 2\pi[$ 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.

Table 1 Main symbols used in the algorithm.

Symb.	*Description*
Q	Planar environment.
$S(v)$	Sensor footprint from pose v.
S_k	Sensor footprint of the $k-$th perception.
∂S_k	Oriented boundary of S_k.
$\mathcal{L}_k$	Free (non-obstacle) boundary of ∂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. $\mathcal{F}_k = \mathcal{F}_{k-1}^{\text{Ext}} \cup \mathcal{L}_k^{\text{Ext}}$.
$\mathcal{F}_{k-1}^{\text{Ext}}$	$\mathcal{F}_{k-1} \backslash \text{closure}(S_k)$.
$\mathcal{L}_k^{\text{Ext}}$	$\mathcal{L}_k \backslash \text{interior}(I_{k-1})$.
V_k	Set of viewpoints at the $k-$th step.

Follower: Each follower is assigned to passively follow a frontier-guard, and this assignment can change as the algorithm progresses.

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

Centralized Clearing Algorithm

Initialize one robot as a frontier-guard, the rest as followers. Then:

```
1: for each $\{S_k, \partial S_k, \mathcal{L}_k\}$ received do
2:     Compute $\mathcal{F}_k$ from $\mathcal{F}_{k-1}$ and $\{S_k, \partial S_k, \mathcal{L}_k\}$ as detailed in Sec. 3.1.
3:     Compute the next set of viewpoints $V_{k+1}$ as detailed in Sec. 3.2.
4:     Assign each $v \in V_{k+1}$ to a nearby searcher and set the searcher to be a frontier-guard.
5:     Assign remaining searchers a frontier-guard to follow.
6:     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.
```

For the reader's convenience we describe the algorithm in detail. At the beginning, 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 a starting viewpoint. All other robots are set as followers of this guard. The initial frontier-guard then records the first perception $\{S_1, \partial S_1, \mathcal{L}_1\}$, 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 ∂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 $S_k \backslash I_{k-1}$ at the instant in which the k-th perception is performed, or (2) by crossing $\mathcal{F}_k$. In the 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. After receiving a new perception the global frontier is updated (Step 2). Next the determination of a new set of viewpoints to cover and expand the frontier is performed (Step 3). After that, searchers are assigned roles as guards or followers and dispatched to their respective target positions (Step 4).

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:

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 $\text{Area}(I_{k+1}) \geq \text{Area}(I_k) + \epsilon$ for some $\epsilon > 0$ except for at most finite steps.

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 $n \geq \max_k |V_k|$, the entire environment Q is cleared and every evader in Q is detected in finite time.*

Proof. The expansion property in Defintion 1 ensures that there will be a time step k_f where $\mathcal{F}_{k_f} = \emptyset$, meaning that ∂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 $S_{k_e} \backslash I_{k_e-1}$ at time $k_e \in \{1, \ldots, k_f\}$, or (2) crosses $\mathcal{F}_{k_{e-1}}$ during the time interval $[k_{e-1}, k_e]$ for $k_e \in \{2, \ldots, k_f\}$. 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. □

In the rest of this Section we describe how to implement the frontier update and how to plan viewpoints (Steps 2 and 3 of the centralized clearing algorithm, respectively). The path-planning in Step 6 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 Global Frontier without a Map of the Environment

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 $\{S_k, \partial S_k, \mathcal{L}_k\}$.

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 ∂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 ∂S_k. Let $\mathcal{L}_k^\star$ 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 $\bar{\mathcal{L}}_k^\star$ 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 $p \in \mathcal{L}_k^\star$ as internal or not
2: Classify the ends of each $\ell \in \mathcal{L}_k$
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

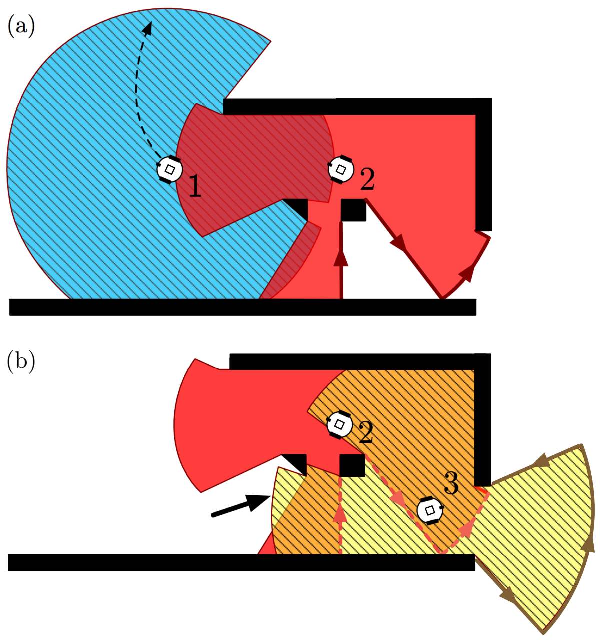


Fig. 2 (a) After robots 1 and 2 have classified their frontiers, robot 1 moves to a new position. Once robot 1 has moved and recorded a new perception, its prior perception is no longer stored by the robot team. (b) When robot 3 arrives and records $\{S_k, \partial S_k, \mathcal{L}_k\}$ (striped yellow), 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 ∂S_k and robot 2's oriented frontier segments (dashed red)

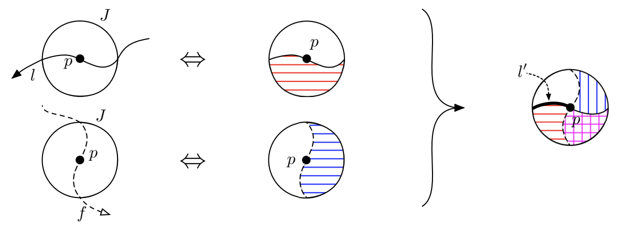
$\mathcal{L}_k^\star$ while the exterior ones belong to $\bar{\mathcal{L}}_k^\star$. The following crucial result states that an arc in $\mathcal{L}_k$ can only switch between the interior and exterior of I_{k-1} at an intersection point in $\mathcal{L}_k^\star$.

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

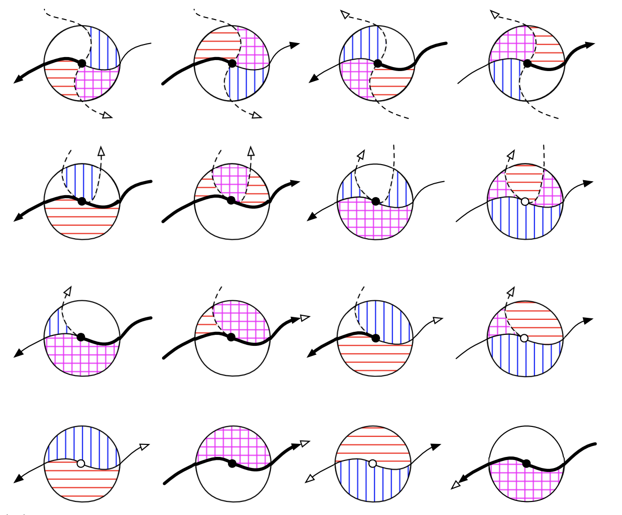
Proof. Since ℓ is in Q, it cannot cross the obstacle boundary of ∂I_{k-1}. Therefore, if ℓ does not intersect $\mathcal{F}_{k-1}$, then it does not cross ∂I_{k-1}. □

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

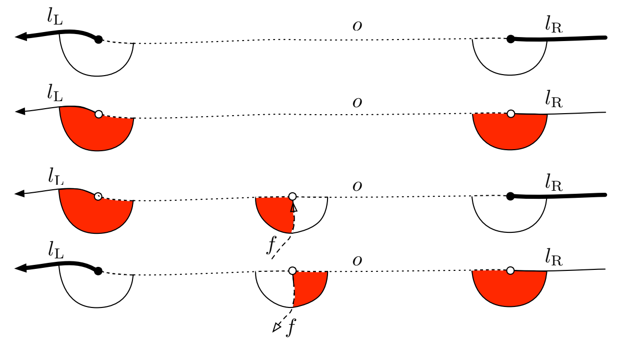
The second step of the method is to classify the ends of each arc $\ell \in \mathcal{L}_k$ in the neighborhood of the endpoints of the adjacent obstacle arcs. These neighborhoods can be classified using the following Lemma.



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



(b) 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 ℓ is drawn solid, while f is dashed. Each row shows a different intersection type, with columns for the various reciprocal orientations of ℓ and f. The first row shows isolated crossings, the second shows isolated tangents, the third shows joinings, and the fourth row shows segments where ℓ and f overlap. The bold portions of ℓ belong to $\mathcal{L}_k^{\text{Ext}}$



(c) 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

Fig. 3 Detailed explanation of frontier classification for (a)-(b) arc crossings and (c) intersections on obstacles

Lemma 2 (Obstacle Arc Classification) *Let o denote a local obstacle arc of ∂S_k, let ℓ_L and $\ell_R \in \bar{\mathcal{L}}_k^\star$ 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:*

- *If $E_o = \emptyset$, then either ℓ_L and ℓ_R are both internal to I_{k-1} or neither are.*
- *If $E_o \neq \emptyset$, then ℓ_L is internal to I_{k-1} if the closest*[1] *$e \in E_o$ is the beginning of a frontier arc, and not internal otherwise. The opposite holds for ℓ_R.*

Proof. If $E_o = \emptyset$, as shown in the first two cases of Fig. 3(c), then there exists a free arc connecting ℓ_L with ℓ_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. 3(c). Then, there exists a free arc connecting ℓ_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 ℓ_R. ☐

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 $\mathcal{L}_k^\star \neq \emptyset$ 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 $\mathcal{F}_k$.

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 ∂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.

[1] With respect to the distance on the arc o.

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:

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

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 2π. Afterwards, we describe how to adapt the method for (d, ϕ)-searchers. In both cases, the actions of the local viewpoint planning method are as follows.

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 This viewpoint planner 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 ℓ_{Rel} is comprised of straight radial segments and circular segments with radius d. The possible configurations are: single radial; single curved; curved with radial on one side; or curved with radial segments on both sides (see the examples in Fig. 1 and Fig. 4). 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 $r \in \mathcal{L}_k^{Rel}$ be a radial free arc segment. Let p be the endpoint of r farthest from v' and $\overline{v'p}$ be the line segment between v' and p. If $\operatorname{dist}(v', p) < d$ and $\overline{v'p}$ only intersects ∂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, $\operatorname{dist}(v', v_k) < d$. Combined with the fact that $\operatorname{dist}(v', p) < d$, 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 ∂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 $S(v')$. □

There are two notable consequences of Lemma 3. First, for any ℓ_{Rel} with only a radial segment, one viewpoint is sufficient. Second, for any ℓ_{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 $d_{\min} \in (0, d]$, the minimum distance between v_k and any $v \in V'$. As will become clear, $d_{\min}$ encodes a trade-off in the algorithm: smaller values of $d_{\min}$ can reduce $|V'|$ and thereby reduce the number of searchers required; larger values of $d_{\min}$ 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 ℓ_{Rel} measured counter-clockwise from the right-most frontier point on ℓ_{Rel} to the left-most frontier point on ℓ_{Rel}. A single viewpoint at least $d_{\min}$ from v_k can then cover an angular width of at most $\alpha(d_{\min})$ given by

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

The number of viewpoints η necessary to cover ℓ_{Rel} is determined by the following Lemma.

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

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

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

For $\eta > 1$, the angular width of ℓ_{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 ℓ_{Rel} achieves step 2 of the viewpoint planning method.

After partitioning ℓ_{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 ℓ_{Rel} and also uncovers as much area as possible beyond ℓ_{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 ℓ_{Rel}. For all other configurations, we construct a line through v_k which bisects the curved arc in ℓ_{Rel} assigned to v. We then place v on this bisector at the point in S_k which is closest to the intersection with ℓ_{Rel} and also ensures that both endpoints of the curved arc in ℓ_{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 $\mathcal{L}_k^{\mathrm{Ext}} \in \cup_{v \in V'} S(v)$ for searchers with $\phi = 2\pi$, meaning that the *frontier guarding property* in Definition 1 is satisfied. For $d_{\min}$ 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*.

Lemma 5 (Guaranteed Expansion) *The set of new viewpoints V' produced by the Local Viewpoint Planning Method satisfies the expansion property from Definition 1.*

Proof. Consider a new viewpoint $v \in V'$ inside of S_k and the associated footprint $S(v)$. What we will show is that, except for at most a finite number of steps, there exists some new area $A \in S(v)$ where $\mathrm{Area}\,(A) \geq \epsilon$ and $\mathrm{Area}\,(A \cap I_k) = 0$ for some small $\epsilon > 0$.

Two properties allow us to determine values for ϵ. We have specified that v will be at least $d_{\min}$ from v_k. Let ℓ_v be the free arc segment assigned to viewpoint v and let r_e be the smallest possible diameter of any evader the team will be asked to detect. Then, we know that the length of ℓ_v is at least r_e or it can be ignored. In the case where ℓ_v is either a radial or curved segment, then a value for ϵ assuming that $d_{\min}$ and r_e are small is $d_{\min} r_e$. The case were ℓ_v is mixed is more intricate, but a similar lower bound can be found.

Finally, the only circumstances in which $\mathrm{Area}\,(A)$ might be less than ϵ occur when either there is a finite-sized obstacle or a finite-length portion of $\mathcal{F}$ which reduces the size of A. These can only occur a finite number of times, so the statement holds. $\square$

We have described a viewpoint planning method which meets the requirements of Definition 1 for searchers with $\phi = 2\pi$. For (d, ϕ)-searchers whose sensors have a field-of-view in $[\pi, 2\pi)$, the above method is optimal in the number of viewpoints required to cover ℓ_{Rel} when it 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 needed. We instead propose the following geometric method.

Consider the case when ℓ_{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 ℓ_{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 ℓ_{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. Third, the distributed algorithm must work with only 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 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 covered 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 would lie in the footprint of the previous one.

Updates to the global frontier in the pairwise frontier update method are based on current relative poses of nearby searchers, not on absolute poses. Therefore, the distributed clearing algorithm can continue clearing an environment even if the searchers cannot determine where they are relative to where they started. Also 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 by either the guard itself or by a follower. Since the new viewpoint is in S_k, 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 to locate a leader to follow.

The *distributed clearing algorithm* 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:

`UpdateNeighFrontier & Frontier`: perform the pairwise frontier update method in Section 4.1.

`ViewPointPlan`: 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`: 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.

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

Figure 4 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 m and a field-of-view of 240°. 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).

Distributed Clearing Algorithm

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:

Procedure `Expand`

```
Data: frontier,path
foreach follower in followers do
    Send(follower, "follow",path);
Move(path);
$\{S, \partial S, \mathcal{L}\}$ ← Perceive();
neighFront ← UpdateNeighFrontier();
frontier ← Frontier($\{S, \partial S, \mathcal{L}\}$,frontier,neighFront);
DoBehavior("Frontier-Guard",$S$,frontier);
```

Procedure `Frontier-Guard`

```
Data: $S$,frontier
if frontier is empty then
    Send(followers, "wander");
    DoBehavior("Wander");
(bestVP,NumVPs) ← ViewPointPlan($S$,frontier);
path ← PathToViewPoint($S$,bestVP);
if NumVPs == 1 then
    DoBehavior("Expand",frontier,path);
else
    if followers has at least one follower then
        follower ← PopFollower(followers);
        Send(follower, "expand",path);
        while follower is expanding do
            Sleep();
    else
        while no new neighbor and no followers do
            Sleep();
    DoBehavior("Frontier-Guard",$S$,frontier);
```

Procedure `Follow`

```
Receive(Leader,message,path);
switch message do
    case "follow"
        Move(path);
    case "expand"
        DoBehavior("Expand",$\emptyset$,path);
    case "wander"
        DoBehavior("Wander");
```

Procedure `Wander`

```
SearchForLeader();
if leader found then
    DoBehavior("Follow");
if all searchers wandering then
    exit
```

The first panel of Fig. 4 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.

5 Theoretical Analysis

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.

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.

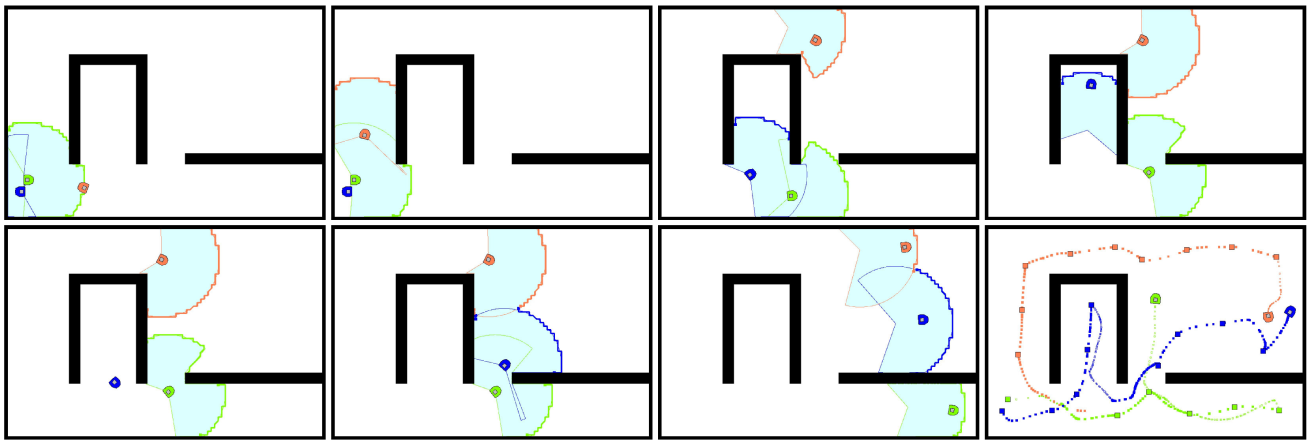


Fig. 4 Simulation of three (d, ϕ)-searchers clearing an environment. Recorded perceptions are shown in a light blue, with frontiers shown with bold lines in the color of the frontier guard who owns them. The trajectories of the robots are shown in the final panel, with large squares for viewpoints.

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 Time and Memory Complexity

The computational requirements of the four main subroutines of the distributed clearing algorithm are as follows. An important innovation of this works is the pairwise frontier merging method, which requires only $\mathcal{O}(2|\partial S|)$ memory and $\mathcal{O}\left(|\partial S|^2\right)$ time to find intersections and classify the local frontier (where ∂S is the boundary of a sensor footprint). 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. 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.

Therefore, we have the following result which satisfies the memory assumption in Section 2.2. Notice that this statement is *per robot*: the frontier-guarding property ensures that the team of searchers will divide the global frontier $\mathcal{F}$ into finite sized pieces.

Lemma 6 (Constant Memory per Robot) *The distributed clearing algorithm requires each searcher have an amount of memory proportional to that required to store two perceptions.*

5.4 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 provided that the number of searchers is known in advance.

5.5 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

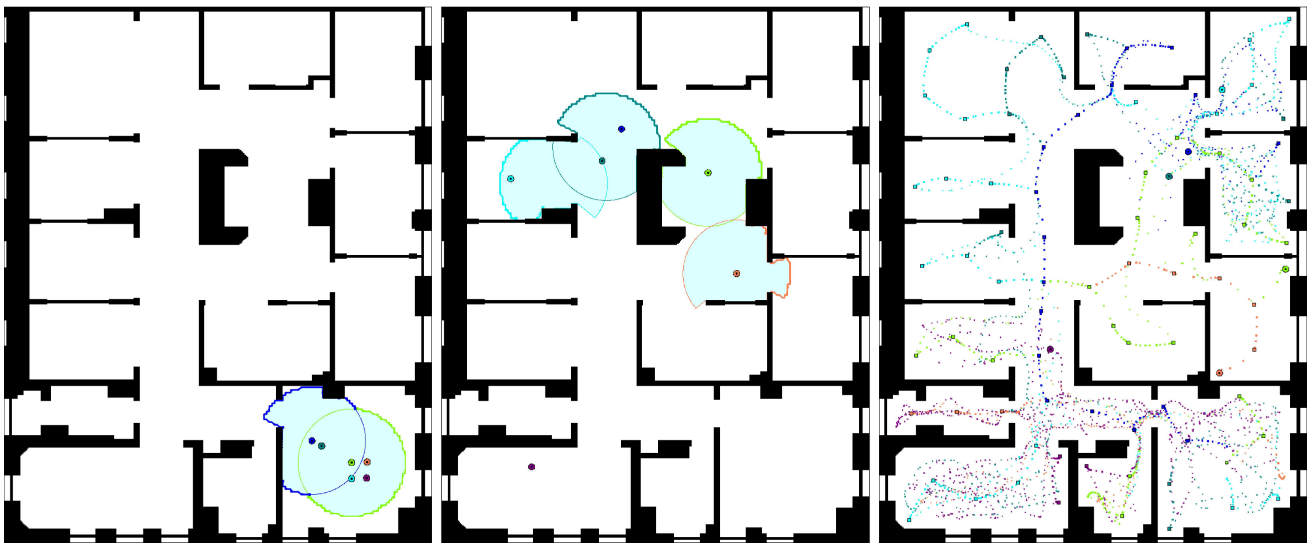


Fig. 5 Three screenshots from a simulation of six d-searchers clearing a portion of a hospital wing. The paths of the agents are shown at right, with all viewpoints drawn with larger squares

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 Experimental Results & Numerical Analysis

To demonstrate the utility of the proposed distributed clearing algorithm, we implemented it using the open-source Multirobot Integration Platform (MIP) (Franchi and Stegagno, 2009) 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 cm resolution[2], with oriented frontier arcs handled as ordered sequences of cells. Each robot stores only its most recent perception and its local frontier.

[2] Such a discretization of local space is useful and common in practice, the resolution of the local grid should be chosen based on the minimum desired detectable evader size.

6.1 Hospital Wing Simulation

Figure 5 presents a simulation in a complex environment modeled after part of a wing of a hospital. A movie version is available in Online Resource 2. The environment is 16 m wide and 20 m tall, with a number of small patient rooms around a central desk, as well as a few other rooms at the bottom of the map.

Six simulated d-searchers with $d = 2.0$ 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 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,

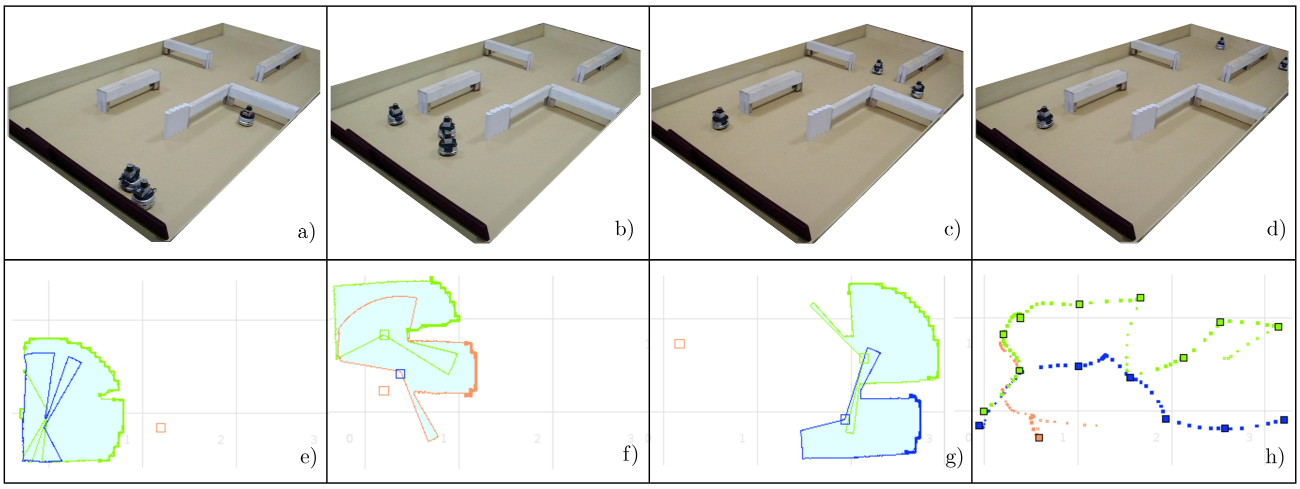


Fig. 6 Four phases of an experiment with three Khepera III robots with Hukuyo URG-04LX laser sensors. One of the robots simulates a motor fault (b,f) which forces the others to complete the task by themselves (d,h)

simple paths taken by searchers executing repeated expansions.

6.2 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° and a range we artificially limit to 0.8 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. 6, 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.

6.3 Area Cleared in Empty Space

In this section we use Monte Carlo simulation to study how the area cleared in an obstacle-free environment changes with the number of robots available. 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 in the absense of obstacles by n d-searchers occurs when the searchers are at the vertices of a n-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 $\frac{nd^2}{\tan\left(\frac{\pi}{n}\right)} + \frac{(n+2)\pi d^2}{2}$. We set $d = 1.0$ m, meaning the limit on the area 12 searchers can clear is 66.8 m.

We conducted 100 simulations for 3 through 12 robots. 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. 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. When the first guard records the initial perception, it has a frontier arc with angular width 2π. We divide followers evenly such that a third of the available robots will end up at each of the three viewpoints needed to expand the initial frontier.

The results of our simulations are shown in Fig. 7. With three searchers the Distributed Clearing Algorithm consistently cleared 95.2% of maximum possible area. This efficiency dipped to 85.2% with six searchers, then increased to 96.8% and 90.6% on average with with 9 and 12 searchers, respectively. A video of an example trial with 12 robots which cleared 96.2% of the optimal area is available as Online Resource 4. The variability in the area cleared with six or fewer searchers is mini-

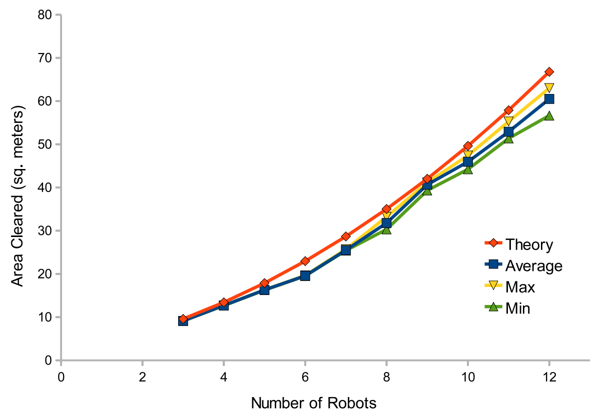


Fig. 7 Comparison of average, maximum, and minimum area cleared in 100 simulations for different numbers of robots to the theoretical limit.

mal because the sequence of expansions is independent of the relative timing of when robots record their perceptions. With six searchers, two go to each of the three viewpoints needed to expand the initial circular frontier. Each of these pairs will then split the next two viewpoints needed to expand their local frontier, and the team will then reach an equilibrium. With eight searchers, the number of expansions increases from 9 to 17 or 18 and these later expansions influence each other by sometimes clearing a significant portion of a neighboring guard's frontier.

These numerical results demonstrate that expansion sequence generated by the Local Viewpoint Planning Method can produce efficient global results without global information even for large numbers of searchers.

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 work 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, Räcke H, Sivadasan N, Sohler C, Vöcking 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

Censi A (2008) An ICP variant using a point-to-line metric. In: 2008 IEEE Int. Conf. on Robotics and Automation, Pasadena, CA, pp 19–25

Durham JW, Bullo F (2008) Smooth nearness-diagram navigation. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Nice, France, pp 690–695

Durham JW, Franchi A, Bullo F (2010) Distributed pursuit-evasion with limited-visibility sensor via frontier-based exploration. In: 2010 IEEE Int. Conf. on Robotics and Automation, Anchorage, AK, pp 3562–3568

Franchi A, Stegagno P (2009) 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 2.13

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

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, Matarić 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

Yamauchi B (1998) Frontier-based exploration using multiple robots. In: 2nd Int. Conf. on Autonomous Agents, Minneapolis, MN, pp 47–53