Pursuit-Evasion in an Unknown Environment Using Gap Navigation Graphs

Luis Guilamo, Benjamin Tovar and Steven M. LaValle
Dept. of Computer Science
University of Illinois
Urbana, IL 61801, USA
{lguilamo, btovar, lavalle}@uiuc.edu

***Abstract*—**
In this paper we present an online algorithm for pursuit-evasion in a unknown simply connected environment, for one pursuer that has minimal sensing and carries a set of stationary sentries that it can drop off and pick up during the pursuit. In our sensing model, the pursuer is only able to detect discontinuities in depth information (gaps), and it is able to find all of the evaders without any explicit localization or geometric information, by using a Gap Navigation Graph. The strategy is based on growing an evader-free region, by reading "exploration" schedules from the Gap Navigation Graph, that is constructed online. We prove that a pursuer with $k + 1$ sentries can clear any environment that could be cleared by k pursuers using the algorithm in [7], which required a complete map and perfect sensing.

I. Introduction

Imagine the following setting: a robotic law enforcer needs to find all suspects inside a building floor from which it does not have a map. The sensors capabilities of this robot are limited; it does not have a compass, and the only tool it has for navigation, a range sensor, gives unreliable information. To make things complicated, of course the suspects are trying to hide from our robotic police. In this paper we propose a strategy for solving this kind of online pursuit-evasion scenarios, using minimal sensing.

The pursuit-evasion problem was treated originally in a game theoretic framework [3], [9]. Pursuit-evasion in graphs has been studied extensively [1], [6], [10], [13], [17], [21]. There have been important developments in the study of pursuit-evasion for mobile robots in the plane in recent years. This problem was introduced in [24], as a dynamic version of the art-gallery problem. A complete algorithm for polygons searchable by one pursuer was presented in [7], in which it was also shown that determining the optimal number of pursuers is NP-hard. Some solutions for particular characteristics of the pursuers have been found. For example, pursuers with a single flashlight [16], or a group of pursuers that are always mutually visible [5], [27]. In [20], it was proved that any polygon searchable by a pursuer having omnidirectional view, is also searchable by a pursuer with two flashlights, which leads to a quadratic-time algorithm.

Until very recently, the availability of a map of the environment was assumed. In [22] an online version of the problem is presented, in which the required sensing capabilities of the pursuer are minimal, and a representation of the environment is not available. In [11], a six-state automaton is designed with the same capabilities as a pursuer with a single flashlight, following the environment's boundary. Our paper uses the same sensing model as [22]. In that work, a strategy was given that enables a single pursuer to clear any polygon that could be cleared by the pursuer with perfect sensing. However, the motions are extremely inefficient and require careful feedback control to prevent accidental loss of previous work. Although the theoretical results are interesting, the approach is impractical for real mobile robot systems.

Our approach is based on the data structure presented for navigation in [26]. Because this approach produces optimal motions, the resulting strategies are very efficient. Detecting discontinuities in depth information and their topological changes, a data structure is built that encodes the same paths as the bitangent graph[1][19]. These discontinuities are called *gaps*, and the structure is referred to here as a *Gap Navigation Graph* (GNG). These critical events consist of appearances, disappearances, splits and merges of gaps. The critical events are used to described the changes of *cleared* regions, where evaders are known not to be, and *contaminated* regions, where an evader may be hidden. The minimalistic sensing and robot capabilities approach is inspired after the bug algorithm framework [4], [12], [23].

By adding information state labels to the GNG, this

[1]In the case of polygons, the bitangent graph has the edges of the visibility graph that are bitangent at two points of the polygon's boundary, and that are totally contained inside the polygon. Enough information is encoded in the bitangent graph to compute shortest paths. The bitangent graph is also known as the *reduced visibility graph*[14].

paper presents an online strategy that finds all of the evaders, if the environment is searchable with one pursuer following paths in the bitangent graph. If the pursuer cannot solve the problem, then it uses portable sensors called sentries to help it guard places while it searches. By using this approach, we show that a pursuer with $k+1$ sentries can clear any environment that could be cleared by k pursuers using the algorithm in [7], which required a complete map and perfect sensing. Furthermore, the resulting solutions are very efficient, which makes the method practical for applications.

II. Problem Definition

One robot, called the *pursuer*, is required to visually locate one or more *evaders*. If the pursuer cannot complete the task alone, it can place guards, or *sentries*, as needed. To reduce the number of sentries that must be carried by the pursuer, the sentries may be reused. The sentry is an immobile observer, which the pursuer can place at any location, or retrieve by "touching". The pursuer and the evaders move in a simply-connected, open subset, R, of $\mathbb{R}^2$. The boundary of R is a simple, closed, piecewise-smooth curve (with a finite number of non-smooth points).

Let $e_i(t) \in R$ denote the position of the i^{th} evader at time $t \geq 0$. It is assumed that $e_i : [0, \infty) \rightarrow R$ is a continuous function, and the evaders can move arbitrarily fast. Let $p(t) \in R$ denote the position of the pursuer at time $t \geq 0$. The pursuer also moves continuously. Similarly, let $s_j(t) \in R$ denote the position of the j^{th} sentry.

For any $x \in R$, let $V(x) \subset R$ denote the set of all $y \in R$ such that the line segment that joins x and y does not intersect the boundary of R. Let $V(x)$ be the *visibility region* of x. If at any time t, either $e_i(t) \in V(p(t))$ or $e_i(t) \in V(s_j(t))$ for some j, then the i^{th} evader is detected. Any subset of R that might contain an evader is referred to as a *contaminated region*. Any region that is guaranteed not to contain an evader is called a *cleared region*. If a cleared region becomes contaminated again, it is referred to as *recontaminated*. For convenience, when an evader is detected, it is assumed to be eliminated.

In addition to the sensor that detects evaders, the pursuer has a sensor that tracks the discontinuities in depth information. Each of these discontinuities is referred to as a *gap*, and the sensor is called a *gap sensor* (Figure 1). Each gap corresponds to a connected subset of the environment that is not visible by the pursuer. The length of each gap, its distance, and its exact angular position are assumed unknown, but it is assumed that the gap sensor is able to track and record the disappearance, appearance, merging, and splitting of gaps. These changes in the gaps are called the *gap critical events*. Information such as exact geometric measurements (i.e., distances and angles) is not recorded; only the gap ordering as they appear in the gap sensor is maintained. Thus, no compass is needed.

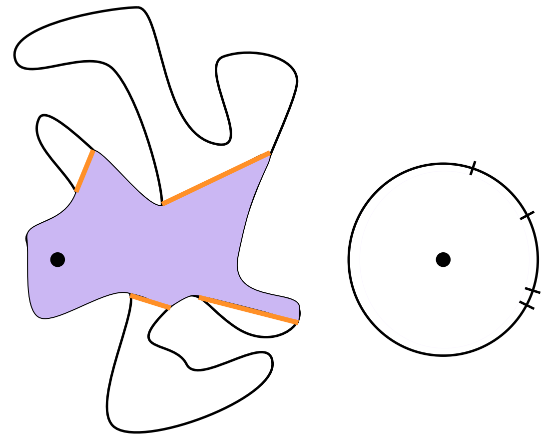

Fig. 1. How the environment appears to the pursuer, which is the black dot in the environment on the left. On the right, the gap sensor shows the location of the discontinuities in the depth information. No metric information (i.e., distances and angles) is reported by the gap sensor.

Using this model, the pursuer is placed into some unknown region R. The task is to design a motion strategy that detects (or eliminates) all of the evaders, while trying to minimize the total number of needed sentries. This will be accomplished by building on a Gap Navigation Graph, which is discussed next.

III. Gap Navigation Graphs (GNGs)

In this section we give a brief description of GNGs, which were introduced in [26], [25] as a means to navigate in unknown planar environments. The motions produced are optimal in simply-connected environments; otherwise, they are locally-optimal. The approach was successfully demonstrated on a mobile robot platform.

In the simply-connected case, the GNG is a tree, hereafter referred to as T_g. Usually the environment is unknown and the GNG is constructed online. The root of T_g moves along with the pursuer. Each child of the root represents a gap that is currently visible, and the gaps are maintained in circular order. In T_g, we use the terms gaps and nodes interchangeably because each node encodes a gap.

As the pursuer moves, critical events are triggered. There are appearances or disappearances of gaps, which occur when the pursuer crosses inflections, and splitting or merging of gaps, which occur when the pursuer crosses bitangents [15]. As events occurs, T_g is updated as follows: if a gap disappears, the corresponding node is removed from T_g. If a gap appears, it is added as a child of the root of T_g in a location that preserves the circular ordering of gaps. Any node that is added in this way is designated

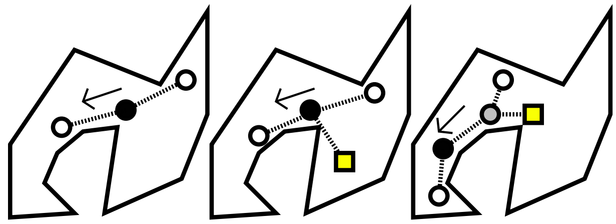

Fig. 2. Encoding of critical events into a Gap Navigation Graph. The black disk denotes the root of T_g and the current position of the pursuer. As the pursuer chases the gap on the left, a gap appearance, and a gap merge is triggered. The structure of T_g is updated accordingly.

as a *primitive node*, which indicates that a portion of the environment that was once visible is now occluded. If a gap splits, then the corresponding child of the root will be replaced with two children. If two gaps merge, the two corresponding children of the root become the children of a new node, d, and d becomes a child of the root (see Figure 2). Merging can only occur between a pair of gaps that are adjacent in the circular ordering produced by the gap sensor. We also make a general position assumption, which is that no two critical events can occur at the same time. For example, in one time instant, three gaps cannot merge into one.

As shown in [26], a sequence of nodes from the root of T_g to a leaf define a sequence of gaps, that if chased, follows a path in the bitangent graph. *Chasing* a gap means the pursuer moves toward the gap, until the gap either splits or disappears. In a real robotic setting a robust navigation system following discontinuities may be implemented, as the one presented in [18],

We next consider augmenting GNGs to handle pursuit-evasion. Since a gap "hides" some region of the environment from the pursuer, it is possible to label each gap as *cleared*, *contaminated* or *recontaminated*. A gap is said to be cleared if it is not possible that an evader could be in the region hidden by the gap; otherwise, the gap is said to be contaminated. A recontaminated gap is a gap once labeled as cleared, but that became contaminated once again.

Initially, all of the gaps in T_g are labeled as contaminated. If a gap appears, it is labeled as cleared, since if an evader is behind the gap, the pursuer would already have detected it. A cleared gap is recontaminated if it merges with a contaminated or a recontaminated gap. If T_g has at least one node that is not cleared, it is labeled as contaminated; otherwise, it is said that T_g is cleared. Solving the pursuit-evasion problem is equivalent to clear each node of T_g.

IV. Preliminaries

As presented in Section II, the pursuer is required to clear an unknown environment using the gap sensor. The main idea is to grow and preserve a connected sequence of cleared gaps. Consider a sequence of cleared adjacent gaps $C = (\alpha_1, ..., \alpha_n)$, in which α_i appears before α_j for any $i < j$ in the gap sensor.

Lemma 1: After one critical gap event, recontamination can only occur on an end of the sequence (thereby reducing the length of C).

Proof: Merges happen only between adjacent gaps. Since $(\alpha_1, ..., \alpha_n)$ are cleared, if a merge happens within this sequence no recontamination will occur. However, α_1 and α_n may merge with a gap outside of this sequence, which allows recontaminations. ■

This means that recontamination cannot occur in the interior of C unless all of the gaps that were in C become recontaminated. Using Lemma 1, one can devise a strategy such that gaps are added to a single sequence of cleared adjacent gaps until all of the detected gaps are cleared. Intuitively, if more than one sequence of cleared gaps needs to be maintained for solving the task, then more than two flashlights would be needed in general, which would contradict [20]. We make a parallel between maintaining a single sequence of cleared adjacent gaps and the *left invariant property* presented in [11]. If at some point the cleared sequence becomes empty, a new one can be started from a different sequence of gaps. Maintaining the cleared sequence is equivalent to maintaining the left invariant property explicitly.

The structure of merges gives a constraint on the possible labeling of the T_g leaves. Let a branch of T_g be defined as the subtree formed by a root's child and its descendants. The labeling of every branch has the following structure:

Lemma 2: If a branch has a cleared node, then all of the nodes of that branch are cleared. Otherwise, the branch may only consist of contaminated and recontaminated nodes.

Proof: Recontamination propagates through the merges. If a cleared node merges with a recontaminated node then it and all of its descendants become recontaminated. It follows that a cleared node may exist in a branch only if all of the merges in that branch are between cleared nodes. ■

The strategy described in Section V is based on Lemmas 1 and 2.

V. Pursuit Strategy

As the pursuer moves in R, gap critical events are triggered and updated in T_g. The pursuer tries to find a sequence of movements that produce the gap critical events sufficient to clear T_g. Under the pursuer model, the movements are restricted to the chasing of gaps. No previous knowledge of T_g or of the environment is assumed, and T_g itself should be constructed online.

Section IV introduced the idea of the maintenance of a sequence, C, of cleared gaps. We require that C only contains children of the root of T_g. At the beginning of the exploration C is empty, since T_g is unknown, and all of their nodes are contaminated. When the first child of the root appears as cleared, it is included in C. Subsequent cleared children are added to C only if they are adjacent (in the gap sensor ordering) to some element in C. A node that becomes recontaminated is removed from C. If an element of C merges, splits or disappears, C is updated accordingly (two elements are replaced with one, one element is replaced with two, or an element is removed, respectively).

A *schedule* is defined as a sequence of gaps that must be chased from beginning to end, in the order they appear in the sequence. A schedule can be associated with a recontaminated or contaminated node in T_g to guarantee that the region it encodes is cleared. The strategy reduces to the generation of the necessary schedules to clear T_g. By Lemma 1, not all possible schedules are useful for the pursuit strategy, since cleared gaps can only be added to C if they are adjacent to some element already in C. To describe an admissible schedule, we introduce the contributor set of a node. For a primitive recontaminated gap $\alpha \in T_g$, the *contributor set* $\kappa(\alpha)$ is defined as the set of all leaves $v \in T_g$ that are in the same branch as α. A schedule for α is called *local* if the pursuer chases first α, and then each element of the contributor set, such that each gap chased is adjacent to the previous one, recursively. Note that a local schedule can easily be obtained from T_g (see Figure 3). The local schedule structure follows the order in which *essential cuts* are explored with *s-triples*, in [20]. In s-triples, the middle element is explored first and then the neighbors. The difference is that a gap may not encode an essential cut, in which case the order is defined recursively.

As the pursuer moves, several recontaminations may occur, and for each of them a local schedule is computed. These schedules are kept in the *schedule list*, SL. When a new schedule is computed, it is inserted at the front of SL. An interesting consequence of Lemma 1 is that the last recontamination will be always adjacent to C. This means that schedules should be read from the front of the list since this will guarantee that gaps cleared will be adjacent to C. Intuitively, by following these schedules, the pursuer is 'exploring' if it is possible to keep two adjacent gaps clear at the same time. If a recontamination occurs, this forces the pursuer to clear the recontaminations and their respective contributor sets.

Before adding a local schedule to SL, it may be modified as follows. Suppose that a schedule for α is already in SL, and a schedule for β is computed. Let $sc(\alpha)$ and $sc(\beta)$ denote the schedules for α and β respectively. If $sc(\alpha) \cap sc(\beta) \neq \emptyset$, then $sc_v(\beta) \leftarrow sc(\beta) - sc(\alpha)$. This modified schedule is called a *valid schedule* for β. There are two reasons why the valid schedules are needed. The first one is that a valid schedule prevents the pursuer from chasing cleared gaps unnecessarily, since it will have to follow twice the gaps repeated in the local schedules. The second reason, and the most important, is that if this reduced valid schedule already causes a recontamination, this new recontamination should be cleared before clearing contributors of recontaminations deeper in SL; otherwise, it is not guaranteed that cleared gaps will be adjacent to C.

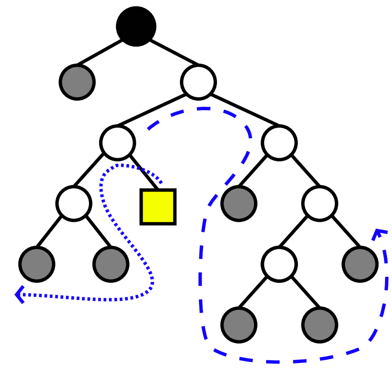

Fig. 3. Reading local schedules from T_g. The black disk represents the root of T_g, and the square denotes the recontamination for which the local schedule will be obtained. By following adjacencies, leaves are added to the schedule first following the dotted line, and then the dashed line, as shown.

The complete strategy for a single pursuer is shown in pseudocode in Figure 4. When the search begins, T_g is initialized with the first reading of the gap sensor, and each node added is labeled as contaminated. Since SL is empty, but there are contaminated nodes in T_g, the pursuer will try to clear them by chasing adjacent gaps. When recontaminations occur, the schedules are computed as described, and inserted in SL. At this point, the pursuer begins to follow schedules in SL, until all of the nodes in T_g are cleared. This strategy is listed as the PURSUIT_SINGLE procedure, in Figure 4. At every moment, the function listed as GAP_TRACKING (also in Figure 4), detects gap critical events, and updates SL and C, as new recontaminations occur.

If a single pursuer is not able to find all of the evaders in an environment by itself, it may use one or more *sentries*. Merges between nodes guarded by a sentry do not cause recontaminations. The pursuer may place and pick up sentries as needed. To detect that one sentry is needed, the first time a local schedule is computed for a given gap, this local schedule is kept in a hash

```
PURSUIT_SINGLE
    T_g ← INITIALIZE_TREE
    SL ← ∅, C ← ∅
    while T_g.label is contaminated
        if SL ≠ ∅
            current_sc ← pop SL
            CHASE(current_sc)
        else
            EXPLORE(T_g)
```

```
GAP_TRACKING
    if gap critical event
        UPDATE(T_g), UPDATE(C)
    if α ∈ T_g is recontaminated
        schedule(α) ← COMPUTE_SCH(α, SL)
        INSERT(schedule(α), SL)
```

```
COMPUTE_SCH(α, SL)
    κ(α) ← CONTRIBUTORS(T_g, α)
    local_sc(α) ← {α, κ(α)}
    schedule(α) ← VALID_SCH(local_sc(α), SL)
    return schedule(α)
```

Fig. 4. Pursuit strategy for a single robot. The pursuer follows schedules to clear contaminated and recontaminated nodes in T_g. A schedule is generated every time a recontamination occurs. The pursuer navigation is determined by the PURSUIT_SINGLE procedure. The GAP_TRACKING function runs all of the time, detecting the gap critical events.

table that is indexed by the branches configuration (not the contamination labeling) of T_g. Elements in the hash table are never updated once stored. When a new local schedule is computed, it is compared with the one kept in the hash table for the current configuration of T_g. If the schedules are the same, we claim that no progress has been made, and another sentry is needed. Since the strategy is deterministic, the pursuer will try the same gap sequences to grow the cleared sequence that did not work before. This is equivalent to finding a cycle in the cleaning order as in [20], but without a map. We show in Section VI that in reaching this conclusion, all possible chasing sequences of adjacent gaps were tried.

To place sentries, two straightforward heuristics are used. One subdivides the environment such that the number of bitangents (and thus of merges) in each contaminated region is minimal. The other tries to maximize the number of contaminated regions separated by the sentry. The pursuer presented here can use both by going to the minimum depth or maximum width T_g, respectively. The pursuer can do that if it keeps the previous states of T_g, together with the record of critical events to reach those configurations. When a region is cleared, the pursuer picks up all but the first sentry placed, to clear the next branch. As it is shown in Section VI, if a sentry is placed where the depth of contaminated branches in T_g is minimal, the number of sentries used is asymptotically optimal (it is $O(\log\ m)$, where m is the number of bitangents in the environment).

If the environment is searchable with one pursuer, but not by chasing gaps, an additional strategy allows the use of only two sentries. A sentry can be placed at the moment where the condition for a new sentry is met. The pursuer then tries to clear each of the regions separated by the sentry. As we will prove in Section VI, if more than two of these regions are not searchable by the single pursuer chasing gaps, the environment is not searchable by one pursuer following arbitrary paths. If only one region is not searchable, a second sentry is placed inside this region where the condition for a new sentry is met. The pursuer then clears the branch of T_g, centered at the first sentry, that encodes the path between the two sentries. At this point, the first sentry can be reused to clear the regions separated by the second sentry, and this strategy is repeated, alternating the picking and placing of sentries. If the branch joining the two sentries cannot be cleared, or if there are two regions not searchable by the single pursuer, the first sentry is still picked up and placed where the condition for a new sentry is reached inside of one of the regions separated by the second sentry.

This is repeated until any branch of T_g centered at a sentry is cleared, in which case alternating the picking up of sentries will clear the environment, or more than two regions will be found to be not searchable. When more than two regions cannot be cleared by one pursuer, the two sentries placed are picked up, and a new sentry is placed according to one of the heuristics described before. The pursuer then tries to clear each region separated by this new sentry, assuming it can be cleared with only two additional sentries. This is repeated recursively.

VI. Analysis

We can compare the performance of the online strategy presented here with a strategy that has access to a map of the environment. As proposed in the next theorem, the strategy presented here has the same searching power as one that has a complete knowledge of the environment, for pursuers only capable of following gaps. For space constraints, we only give an overview of the proof.

Theorem 3: If the environment can be cleared by one pursuer that has a map and chooses to move only along bitangents and the boundary of R, then the environment can be cleared by one pursuer that builds and uses the Gap Navigation Graph, instead of an exact map.

Proof Overview: First consider a strategy that has access to a map of environment, for a pursuer with only a gap sensor. The *visibility cell decomposition* [8] can be computed for the environment representation. For each of the visibility cells, the visibility tree can be computed. A visibility tree is a shortest path tree with the root placed in a given visibility cell. The visibility tree and the Gap Navigation Graph, encode the same path information[2], [26].

A graph $G = (V, E)$ can be constructed such that each vertex in V represents a visibility tree, and $(u, v) \in E$ if and only if by following a gap (a edge of the bitangent graph), the tree represented by $u \in V$ can be transformed into the tree represented by $v \in V$. For each node, a *contamination state* is kept, which gives the cleared or contaminated status for each of the children of the root of the respective visibility tree. If following the edge $(u, v) \in E$, one child is added from the tree in u to the tree in v, this child is labeled as cleared. This is equivalent to an appearance event in the GNG. Merges, splits and disappearances of children from one tree to another are cleared or recontaminated as their counterparts in the GNG. At the beginning, all of the nodes of all of the trees are labeled as contaminated.

Based on $G(V, E)$, a search can be performed, similar to the one presented in [7], by maintaining the cleared/contaminated labels, and updating the contamination states as edges in the graph are transversed. The search ends when a contamination status with all of the children labeled as cleared is reached. This algorithm returns the gaps chasing sequence to clear the environment, if such sequence exist. We now show that the pursuit strategy presented in this paper is an online version of the search just described.

For simplicity, assume that the valid schedules in the list SL are composed only by one gap. If $SL = \{(\alpha_1), (\alpha_2), ..., (\alpha_n)\}$, the pursuer first tries to clear α_n (being the last recontamination), and then proceeds to clear α_{n-1}. At this point, if α_n gets recontaminated, it will be cleared a second time, since it is again the last recontamination. The gaps α_{n-1} and α_n would then have been explored in all possible manners. If α_{n-1} and α_n do not get recontaminated, they now belong to C, and the next gap to clear will be α_{n-2}. Now we can repeat the argument, but with C and α_{n-2} (instead of α_j and α_{j-1}). The former elements of C are cleared again recursively, because of the contributors structure. This is done with the updated C and subsequent recontaminations, generating the sequences with all possible combinations of adjacent gaps.

It is shown in [26] that there exists a path between the current position of the robot and any region of the space encoded by a node in the tree. At worst, the strategy presented in this paper will try all possible gap sequence

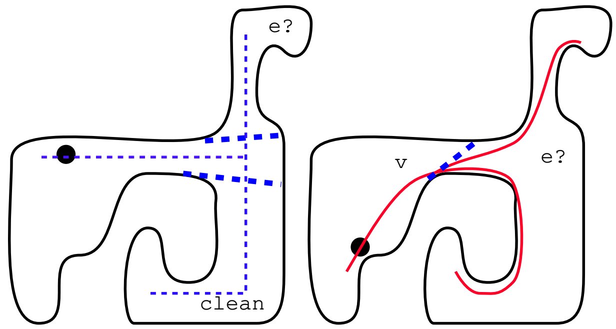

Fig. 5. Not every environment searchable with one robot is searchable with one pursuer following sequences of gaps. A robot following the dashed path can find all of the evaders (left) . The thick paths generated by chasing gaps cause unavoidable recontaminations (right).

combinations. If a solution exists for one pursuer, it cannot lie outside of the set of all possible combinations of adjacent gap sequences. If a solution does not exist, the condition for a new sentry is then fulfilled. ■

An environment may not be searchable by one robot chasing gaps, although it may be searchable by one robot following arbitrary paths (paths unreachable with the GNG). An example of such an environment is presented in Figure 5. It is therefore worth investigating the number of extra robots/sentries required to search such environments.

Theorem 4: If the environment is searchable by one robot, but not necessarily by following paths generated by chasing sequences of gaps, then a pursuer that chases gaps can solve the problem using at most two sentries.

Proof: We use the idea of *separability*, defined in [24]. Let R represent the environment, as defined in Section II. For points x, y and $z \in R$, point x and y are said to be *separable* by x if every path between y and z contains at least one point in $V^2(x)$, in which $V^2(x)$ is the 2-visibility of x. The 2-visibility is defined as $V^2(x) = \cup_{y \in V(x)} V(y)$. Points x, y and z are said to be *mutually non-separable* if no two points out of three are separable by the third. If R is searchable by one pursuer, then no three points in R are mutually non-separable[24].

Assume a sentry is placed the first time the condition for a new sentry is detected, as in Section V. If one branch of T_g, with the root at the sentry, is not searchable by a single pursuer, it has at least two merges (if it has only one, it is easily searchable by the pursuer and the sentry already placed). The 2-visibility of a point encoded in the second level of merges does not include the region encoded by the root of T_g (it can "see" behind the second level of merges, but not the first). Since there are paths from one branch to another that are not separable, then R is not searchable by a single pursuer. If R is searchable by one pursuer not chasing gaps, at most two branches of T_g centered at a sentry will not be searchable by the pursuer following gaps. In this case, the two branches can be cleared as presented in V, by reusing two sentries. ■

Since determining the optimal number of pursuers is NP-hard[7], we can only give a bound in the number of sentries needed in the general case. This bound is presented in the following theorem.

Theorem 5: The number of sentries needed by a pursuer following gaps is $O(\log\ m)$, in which m is the number of bitangents in the environment. This bound is asymptotically optimal.

Proof: Each merge in a branch of T_g encodes one of m bitangents of the environment. If the sentry is placed where T_g has minimum depth, the number of merges in each branch is at most $m/2$, otherwise, there is some configuration of T_g with less depth. Using this argument recursively in each of the regions separated by a sentry, it follows that the number of sentries needed is $O(\log m)$. As shown in [7] for polygonal environments, this bound is asymptotically optimal, and this can be easily extended for regular simple, closed, piecewise-smooth curves. ■

In the special case of polygonal environments, the number of bitangents is $m = \Theta(n^2)$, in which n is the number of vertices. Thus, the number of sentries needed for a polygonal environment is $O(\log n)$.

We now compare the pursuit strategy presented here with other strategies using the same heuristics in the placing of sentries. Two such heuristics were mentioned in Section V. One subdivides the environment such that the number of bitangents is minimized in each contaminated region, while the other tries to maximize the number of regions separated by the sentry. If a strategy with pursuers moving on arbitrary paths using one of these heuristics needs k pursuers, then the pursuit strategy presented here will need at most $k + 1$ sentries and one pursuer. The pursuer can place $k - 1$ sentries following the heuristic, and clear the regions searchable by chasing sequences of gaps. This gives a total of k "observers". The pursuer can then translate the two remaining sentries from one place to another, when a particular region is not searchable by following gaps.

VII. Simulations

We implemented in C++ the strategy proposed for a single pursuer, using a standard desktop PC. We are in progress of extending the computer simulation to include sentries. A map is needed for the simulation, but it is explored by using a simulated gap sensor. Figure 6.(a) shows the sequence of movements the pursuer followed. The initial position of the pursuer is shown with the black disk. Note that the figure only shows the order of how the regions of the environment were visited, but not the actual paths. In Figure 6, the root of the tree denotes the position of the robot, and the tree represents the current state of T_g. Contaminated nodes are denoted with a black circle, and recontaminated nodes are marked with a white circle. Brown (dark grey) areas denote contaminated regions, while green (light grey) regions show cleared areas. A recontamination is shown from Figure 6.(c) to Figure 6.(d), and this is reflected in a merge in the tree. The light grey nodes in the tree on Figure 6.(d) show the schedule to clear this recontamination. We are currently extending the simulation to include sentries.

VIII. Conclusions and Future Work

This paper presented a strategy for pursuit-evasion for a robot with minimal sensing capabilities, and without a map of the environment. By associating the cleared, contaminated, recontaminated labels to discontinuities in depth information (gaps), the pursuer can compute a series of schedules to detect evaders in the environment. If the environment can be search with one robot following paths in the bitangent graph, these schedules will assure the detection of all of the evaders, using a single pursuer; otherwise, only an extra sentry (guard) is required. While understanding that determining the optimal number of pursuers and sentries is a NP-hard problem, we enable a pursuer with $k + 1$ sentries to clear any environment that could be cleared by k pursuers using the algorithm in [7], which required a complete map and perfect sensing.

An interesting direction for future research is the extension of the strategy for multiply-connected environments, by using the GNG for this case [25]. Also, it will be interesting to determine what necessary capabilities should be added to the robot to eliminate the extra sentry for environments that are searchable with one robot.

Acknowledgments

This work is supported in part by ONR Grant N000014-02-1-0488 and NSF-CONACyT Grant 0296126.

References

[1] Micah Adler, Harald Räcke, Naveen Sivadasan, Christian Sohler, and Berthold Vöcking. Randomized pursuit-evasion in graphs. In *29th International Colloquium on Automata, Languages, and Programming*, 2002.

[2] B. Aronov, L. Guibas, M. Teichmann, and L. Zhang. Visibility queries in simple polygons and applications. *Algorithms and Computation, 9th International Symposium, ISAAC '98, Taejon, Korea, December 14-16, 1998, Proceedings*, 1533, 1998.

[3] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Academic Press, London, 1982.

[4] H. Choset and J. Burdick. Sensor based planning, part I: The generalized Voronoi graph. In *IEEE Int. Conf. Robot. & Autom.*, pages 1649–1655, 1995.

[5] Alon Efrat, Leonidas J. Guibas, Sariel Har-Peled, David C. Lin, Joseph S. B. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. In *Symposium on Discrete Algorithms*, pages 927–936, 2000.

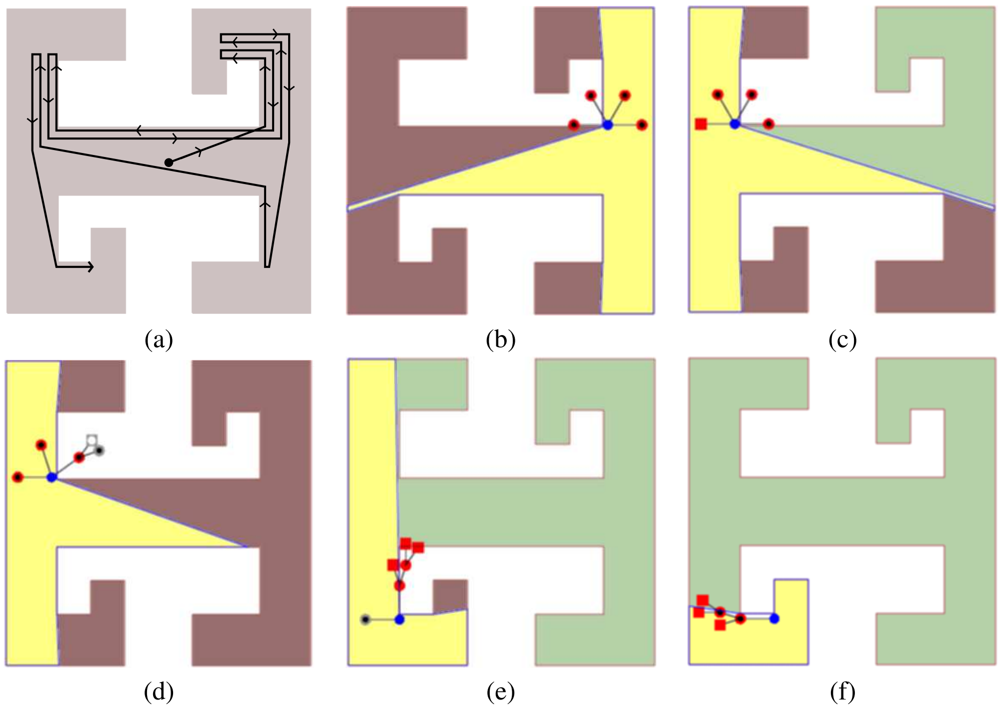

Fig. 6. Simulation results. In (a), the sequence of how the regions are visited in the environment is shown. In (b), (c), (d), (e), and (f), the tree represents the current state of T_g. The root of the tree also represents the position of the robot. Brown (dark grey) areas denote contaminated regions, while green (light gray) areas denote cleared regions. The yellow (almost white) area shows the current visible area of the pursuer. From (c) to (d) a recontamination is shown (details in Section VII).

[6] J. A. Ellis, Ivan Hal Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[7] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999.

[8] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. 1st Workshop on Algorithmic Foundations of Robotics*, pages 269–282. A.K. Peters, Wellesley, MA, 1995.

[9] R. Isaacs. *Differential Games*. Wiley, New York, NY, 1965.

[10] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with limited visibility. In *ACM-SIAM Sympos. Discrete Algorithms*, 2004.

[11] T. Kameda, M. Yamashita, and I. Suzuki. On-line polygon search by a six-state boundary 1-searcher. Technical Report CMPT-TR 2003-07, School of Computing Science, SFU, 2003.

[12] I. Kamon, E. Rivlin, and E. Rimon. Range-sensor based navigation in three dimensions. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[13] A. S. Lapaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, April 1993.

[14] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[15] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–201, April 2001.

[16] S. M. LaValle, B. Simov, and G. Slutzki. An algorithm for searching a polygonal region with a flashlight. *International Journal of Computational Geometry and Applications*, 12(1-2):87–113, 2002.

[17] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, January 1988.

[18] J. Minguez and L. Montano. Nearness diagram navigation (ND): Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, February 2004.

[19] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *1st International Joint Conference on Artificial Intelligence*, 1969.

[20] S.-M. Park, J.-H. Lee, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searcher. Technical Report CS/TR-2001-161, KAIST, Dept. of Computer Science, Korea, January 2001.

[21] T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Application of Graphs*, pages 426–441. Springer-Verlag, Berlin, 1976.

[22] S. Sachs, S. Rajko, and S. M. LaValle. Visibility-based pursuit-evasion in an unknown planar environment. *Submitted to International Journal of Robotics Research*, 2003.

[23] A. M. Shkel and V. J. Lumelsky. Incorporating body dynamics into sensor-based motion planning: The maximum turn strategy. *IEEE Trans. Robot. & Autom.*, 13(6):873–880, December 1997.

[24] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. Computing*, 21(5):863–888, October 1992.

[25] B. Tovar, S. M. LaValle, and R. Murrieta. Locally-optimal navigation in multiply-connected environments without geometric maps. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2003.

[26] B. Tovar, S. M. LaValle, and R. Murrieta. Optimal navigation and object finding without geometric maps or localization. In *Proc. IEEE International Conference on Robotics and Automation*, 2003.

[27] L. H. Tseng, Paul J. Heffernan, and D. T. Lee. Two-guard walkability of simple polygons. *International Journal of Computational Geometry and Applications*, 8(1):85–116, 1998.