Pursuit-Evasion with Fixed Beams

Nicholas M. Stiffler   Jason M. O'Kane

***Abstract*— We introduce a complete algorithm for solving a pursuit-evasion problem in a simply-connected two-dimensional environment, for the case of a single pursuer equipped with fixed beam sensors. The input for our algorithm is an environment and a collection of sensor directions, in which each is capable of line-of-sight detection in a fixed direction. The output is a pursuer motion strategy that ensures the detection of an evader that moves with unbounded speed, or a statement that no such strategy exists. The intuition of the algorithm is to decompose the environment into a collection of convex conservative regions, within which the evader cannot sneak between any pair of adjacent sensors. This decomposition induces a graph we call the *pursuit-evasion graph (PEG)*, such that any correct solution strategy can be expressed as a path through the PEG. For an instance defined by m beams and an environment with n vertices, the algorithm runs in time $O(2^m n^2)$. We implemented the algorithm in simulation and present some computed examples illustrating the algorithm's correctness.**

I. Introduction

This paper considers the problem of planning motions for a mobile robot equipped with a finite collection of single-direction sensors, with the goal of locating an adversarial evader within the line-of-sight of one of those sensors. This problem can viewed as a restricted version of several others in the literature, including the algorithms of Guibas, Latombe, LaValle, Lin, and Motwani [5] (in which the robot has an omnidirectional sensor); Gerkey, Thrun, and Gordon [4] (in which the robot has an angle-bounded but continuous and rotatable field of view); and Kameda, Yamashita, and Suzuki [9] (in which the robot, called a 1-searcher, has a single rotatable beam sensor).

The unique restriction that we consider here is that the directions of the sensor, expressed in world coordinates, are *fixed*. The pursuer robot cannot rotate to aim its sensors in its search for the evader; it must locate the evader using only translations. The new contribution of this paper is a complete and efficient algorithm for solving this fixed-beam pursuit-evasion problem. We also present an implementation of this algorithm, and show computed examples demonstrating its correctness and effectiveness.

Figure 1 shows an example pursuit plan generated by our algorithm. In this instance, the pursuer has four beams, oriented in up, down, left, and right positions.

N. M. Stiffler and J. M. O'Kane are with the Department of Computer Science and Engineering, University of South Carolina, 301 Main St., Columbia, SC 29208, USA. {stifflen, jokane}@cse.sc.edu

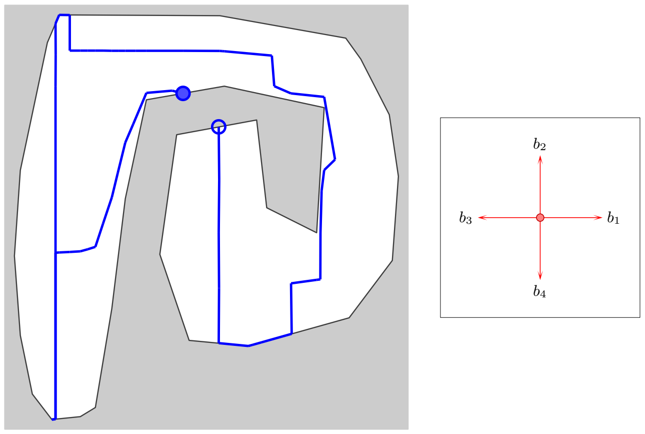

Fig. 1. A pursuit plan (left) computed by our algorithm. The pursuer uses four orthogonal beams (right) to capture the evader, regardless of the evader's path or velocity. The pursuer starts on the bottom boundary of the top-center corridor, and travels first to the left and then to the right.

(Our algorithm works for arbitrary collections of beams, not just orthogonal ones.) Starting from the bottom of the middle section of the spiral, the pursuer travels left to the end, then back around to the center of the spiral. In several cases, the robot moves to the boundary of the environment, which is necessary in this problem to ensure catching any evaders if the area between two beams has non-zero area, then the evader can hide there indefinitely. Our algorithm is complete, in the sense that if a path exists to clear the given environment with the given beams, we are certain to find it. If no such path exists, the algorithm terminates with a failure result.

Our work on this problem is motivated by several related factors. First, and most importantly, it provides another data point for understanding the computational, sensing, and movement requirements that underlie the problem of searching for evaders. There is an obvious connection between the pursuer's sensing and movement capabilities and the existence of a solution. Any instance that can be solved with weak sensors can also be solved with relatively stronger sensors [21]. The relationship between sensing capabilities and the computation needed for planning is less obvious. For example, compared to the existing algorithm for omnidirectional sensing [5], the computation time is *increased* (due to an additional dimension of the underlying C-space) by a restriction to a rotatable range of sensing angle [4], but *decreased* (down to constant memory and constant time to compute the next movement) by a further restriction of that range to a single rotatable direction. Our results, which include an algorithm whose run time is polynomial in the complexity of the environment but exponential in the number of sensors, suggest that the computational difficulty of these kinds of problems is governed not just by the informative value of the sensors, but also by the *complexity* of that sensor model, measured informally by the non-trivial relationships between information that is observable and information that is unobserved.

We also suspect that there may be some direct value to studying restricted versions of planning problems, as an algorithmic tool for solving the original, unrestricted problems. The idea draws inspiration from the Miller-Rabin primality-testing algorithm [17], [26], which employs a probabilistic test that determines, with a known success probability, whether a given number is 'definitely composite' or 'possibly prime'. The algorithm works by iterating this test, until the input integer is demonstrated to be composite, or until its probability of being composite in spite of repeatedly passing the test becomes acceptably small. We are likewise interested in planning algorithms that attack challenging problems by attempting to solve a randomly-generated series of restricted, but efficiently-solvable, related instances. Under the right conditions, we can ensure that if any of those restricted instances has a solution, the solution applies to the original problem as well. The algorithm we propose here is a first step toward applying that strategy to the full omnidirectional visibility-based pursuit evasion problem.

The remainder of this paper is laid out as follows. After reviewing related work (Section II) and formally defining the problem (Section III), we present a complete algorithm (Section IV) along with an implementation and some computed examples (Section V). The paper concludes with a preview of future work (Section VI).

II. Related Work

This section looks at existing literature in the field of pursuit-evasion. Although this paper presents a result for a visibility-based pursuit-evasion problem we discuss the evolution of the pursuit-evasion problem from differential games (Section II-A) to a graph-based formulation (Section II-B) and finally to a geometric formulation (Section II-C).

A. Differential Games

The pursuit-evasion problem was originally posed in the context of differential games [6], [7]. This has led to many different variations of the pursuit-evasion problem. In the lion and man game, a lion tries to capture a man who is trying to escape [20], [30]. In game theory, the homicidal chauffeur is a pursuit evasion problem which pits a slowly moving but highly maneuverable runner against the driver of a vehicle, which is faster but less maneuverable, who is attempting to run him over [7], [28]. Capture bounds for a pursuit-evasion problem that require the pursuer to physically capture the evader suggests the number or pursuers required to satisfy this capture condition exceeds that needed for the visibility-based pursuit-evasion problem [10].

B. Graph-Based Formulation

Pursuit-evasion on a graph can be traced back to the independent work done by Parsons and Petrov. The motivation behind the Parson's problem was the desire for a graphical model to represent the problem of finding an explorer who is lost in a complicated system of dark caves. The idea behind the Parson's problem [23], also known as the edge-searching problem, is to determine a sequence of moves for the pursuers that can detect all intruders in a graph using the least number of robots. A move consists of either placing or removing a robot on a vertex, or sliding it along an edge. A vertex is considered guarded as long as it has at least one robot on it, and any intruder located therein or attempting to pass through will be detected. A sliding move detects any intruder on an edge.

The Parson's problem and some of its results were later independently rediscovered by Petrov [24] using slightly different motivating problems. Petrov's formulation considered the cossacks and the robber game [25] and the princess and the monster problem [7]. Golovach showed that both problems considered an equivalent discrete game on graphs.

There are variations of graph-based pursuit-evasion that consider both edge guarding and node guarding. One such formulation that differs from edge-searching (where searchers move across edges and guard vertices) that has a direct application to robotics is the Graph-Clear problem [13]. Graph-Clear is a pursuit-evasion problem on graphs that models the detection of intruders in an environment by robot teams with limited sensing capabilities.

C. Geometric Formulation

The visibility-based pursuit-evasion problem and the surveillance/tracking problem are various types of pursuit-evasion problems that use a geometric formulation.

1) Visibility-Based Pursuit-Evasion: The first visibility-based pursuit-evasion problem was proposed by Suzuki and Yamashita [34] as an extension of the watchman route problem [2] and is a geometric formulation of the traditional graph-based pursuit-evasion problem. Research on the visibility-based pursuit-evasion problem has produced numerous results for both the single pursuer and multiple pursuer variants of the problem.

2) Single Pursuer Visibility-Based Pursuit-Evasion: There are many interesting results for the single pursuer visibility-based pursuit-evasion problem. A complete solution [5], a randomized solution [8], and an optimal shortest path solution [31] have been found.

The capture condition for the general visibility-based pursuit-evasion problem is defined as having an evader lie within the pursuer's capture region. There has been substantial research focused how the visibility-based pursuit-evasion problem changes when a robot has different capture regions. The k-searcher is a pursuer with k visibility beams [16], [34], the ∞-searcher is a pursuer with omni-directional field of view [5], [22], and the ϕ-searcher is a pursuer whose field-of-view [4] is limited to an angle $\phi \in (0, 2\pi]$. Note that all of these approaches consider evaders with unbounded speed.

Others have studied scenarios where there are additional constraints on the pursuer, such as the case of curved environments [15], an unknown environment [29], a maximum bounded speed for the pursuer [35], or has constraints similar to those of a typical bug algorithm [27].

3) Multiple Pursuer Visibility-Based Pursuit-Evasion: As a result of the problem complexity, there is a wide range of literature with differing techniques attempting to solve the multi-robot visibility-based pursuit-evasion problem [32], [33]. Some recent results involve using some of the pursuers as stationary sentinels while other pursuers continue with the search [12]. Another approach involves maintaining complete coverage of the frontier [3]. There are other variants of the pursuit-evasion problem where the pursuers are teams of unmanned aerial vehicles [11].

4) Surveillance: A problem that is very similar to the pursuit-evasion problem discussed above is that of maintaining visibility of a moving target. This type of surveillance problem where one agent tracks another differs in that the pursuer immediately loses the game if he loses sight of the evader. There have been various strategies employed to maintain visibility of a moving target in environments with obstacles [1], [14], [18], [19].

III. Problem Formulation

We consider a known **environment** $W \subset \mathbb{R}^2$. The boundary of W, denoted ∂W, is a simple polygon with n vertices. We assume that W is a closed set, so that $\partial W \subset W$.

A point robot called a **pursuer** moves along a continuous path through W. We let $p(t) \in W$ denote the pursuer's position at time t. The pursuer's speed is bounded, without loss of generality, by 1. We also model the **evader** as a point moving in W, whose position at time t is denoted $e(t) \in W$. To ensure a true worst-case analysis, we allow the evader to move at any finite speed, as long as its path is continuous.

The pursuer is equipped with a set B of m **beam sensors**, each of which can detect the evader by line-of-sight in a single, fixed direction. We represent these directions as a collection of unit vectors $B = \{b_1, \dots, b_m\}$. These directions remain constant as the pursuer moves; the pursuer cannot rotate them. A beam sensor $b_i \in B$ **detects** the evader at time t if there exists a non-negative scalar a such that $e(t) = p(t) + ab_i$ and the line segment connecting $p(t)$ and $e(t)$ is fully contained in W, that is, if $\overline{p(t)e(t)} \subset W$.

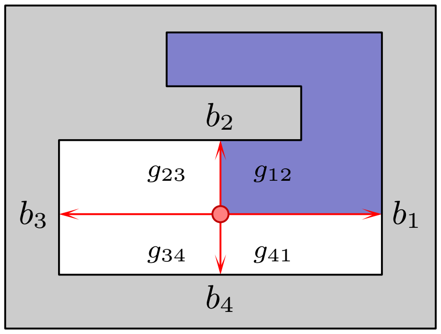

Fig. 2. A robot with four fixed beam sensors. In this example gap g_{12} is shaded green.

The inputs to our algorithm are an environment W, a set of beams $B = \{b_1, \dots, b_m\}$, and a starting pursuer position $p(0) \in W$. The goal is to compute a finite-length path $p : [0, T] \to W$ for the pursuer that guarantees that at least one beam will detect the evader, or report that no such path exists. That is, the algorithm should generate path p starting from the given $p(0)$, and a termination time T, such that for any continuous evader path e, there exists some time $t \le T$ and some beam $b_i \in B$ such that beam b_i detects the evader at time t.

IV. Description of Algorithm

This section describes an algorithm for the pursuit-evasion problem introduced in Section III. Although, at a very high level, the structure of the algorithm follows the same form as existing algorithms for related problems [4], [5], [32], this algorithm differs substantially in its important details. The intuition is to keep track, using a small collection of boolean labels, of which portions of the environment might contain the evader if it has not yet been detected. We partition the environment into a finite set of regions called conservative regions, within each of which the labels remain constant, and track how the labels change as the robot moves between those conservative regions. This induces a graph, through which the algorithm searches for a path from the node representing the initial condition to one in which the evader has certainly been captured. The remainder of this section describes the details.

A. Gaps

In general, the pursuer's m beam sensors divide the environment into a collection of m regions, called **gaps**, that are not currently detectable by any of those beams. More precisely, a **gap** is a maximal path-connected component of the environment that does not cross any of the beams.

Figure 2 shows an example. Note that, if the pursuer is in the interior of W, then each gap includes two beams on its boundary. We write g_{ij} to denote the gap whose boundary includes b_i and b_j.

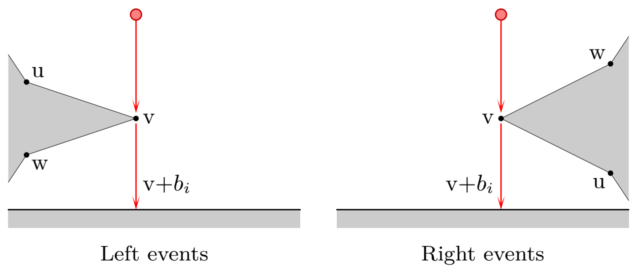

Fig. 3. An illustration to detect when a **Left** event occurs. An illustration to detect when a **Right** event occurs.

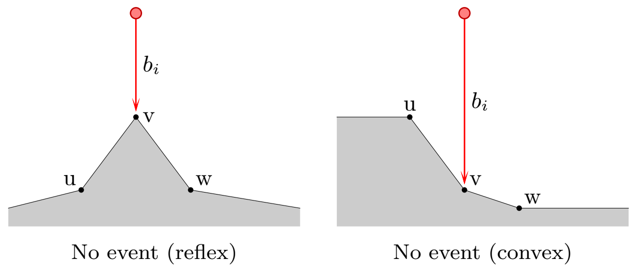

Fig. 4. An illustration that demonstrates when an event does not occur at a reflex vertex. An illustration that demonstrates when an event does not occur at a convex vertex.

The important idea is that the evader, if it has not been captured, is always contained in exactly one gap, in which it can move freely. Although the pursuer does not know the evader's position, it can infer, based on its prior movements, whether an evader could potentially reside within each gap.

A gap g_{ij} is **cleared** at time t if, based on the pursuers' motions up to time t, it is not possible for the evader to be within g_{ij} without having been captured. A gap is **contaminated** if it is not clear. That is, a contaminated gap is one in which the evader may possibly reside. We assign a binary label to each gap corresponding to its cleared/contaminated status. A label of 0 means that the gap is cleared; a label of 1 means that the gap is contaminated.

Notice that, since the evader can move arbitrarily quickly, the pursuer cannot draw any more detailed conclusion about each gap other than its clear/contaminated status; if any part of a gap can contain the evader, then the entire gap is contaminated. As a result, we can encapsulate all of the information available to the pursuer by tracking only the pursuer's current configuration and the current gap labels.

B. Decomposition into Convex Conservative Regions

The algorithm begins by decomposing the environment into a collection of convex conservative regions. A region $R \subset W$ is **conservative** if the gap labels (clear or contaminated) remain unchanged as the pursuer moves within R.

First, we partition the environment into conservative regions by identifying segments in the interior of F at which changes to the gap labels can occur. For a given beam $b_i \in B$, the crucial locations for the pursuer are positions $p(t)$ at which the ray extension $p(t)+ab_i$ within W ends at a vertex v of W. At such points, the distance observed by the beam can change discontinuously, potentially allowing the evader to transit from one gap to another.

There are three distinct cases, of which only two can cause a change in the pursuer's gap labels (Figure 3), and one is safely ignored (Figure 4). We distinguish these cases via clockwise (cw) and counterclockwise (ccw) tests involving the vertex v and its immediate predecessor u and successor w (in clockwise order) along ∂W.

1) A **left** event occurs when the following condition is satisfied:

$$\mathrm{cw}(\mathrm{v}, \mathrm{v}+b_i, \mathrm{u}) \text{ and } \mathrm{cw}(\mathrm{v}, \mathrm{v}+b_i, \mathrm{w}).$$

That is, left events are generated when the boundary curve at v is on the left side of $v + b_i$.

2) A **right** event occurs when the following condition is satisfied:

$$\mathrm{ccw}(\mathrm{v}, \mathrm{v}+b_i, \mathrm{u}) \text{ and } \mathrm{ccw}(\mathrm{v}, \mathrm{v}+b_i, \mathrm{w}).$$

Right events occur when the boundary curve at v is on the right side of $v + b_i$.

3) The case in which u and w are on opposite sides of a beam does not generate any change to the gap labels because b_i changes continuously at this point, regardless of whether v is a convex or reflex vertex.

For each vertex $v \in W$ and $b_i \in B$, the set of pursuer positions that generate such events can be found by extending a ray within W, starting at v, in direction $-b_i$.

If we additionally know the direction (that is, forward or backward) with which the pursuer crosses this critical event, we can classify the event further.

- If the pursuer is moving so that the endpoint of b_i sweeps across $\overline{uv}$ before reaching v, then after crossing v, the beam will **extend** to a new environment edge.
- Conversely, if b_i crosses v in the opposite direction, the beam will **retract** to environment edge $\overline{uv}$.

By performing these ray extensions, the algorithm forms a decomposition of the environment into conservative regions. See Figure 5. We represent this decomposition as a doubly-connected edge list (DCEL). Each interior half-edge in the DCEL is labeled with the event type (left or right; extend or retract) and the generating beam b_i.

Note in particular that, although this decomposition generates regions that are conservative, those conservative regions are not necessarily convex. To enable straightforward generation of a path from a sequence of adjacent regions (which will be the final step of the algorithm; see below), we refine the partition via trapezoidal decomposition, ensuring that every region is convex. Figure 6 shows an example of the final decomposition. Half-edges added at this stage are not labeled with any events.

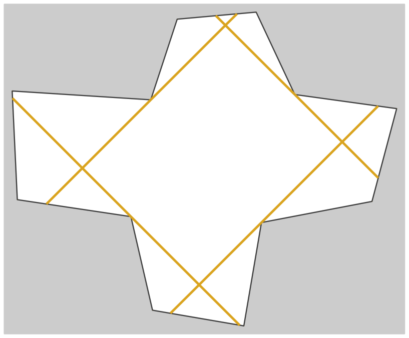

Fig. 5. Decomposition of a simple environment into conservative regions by ray extensions.

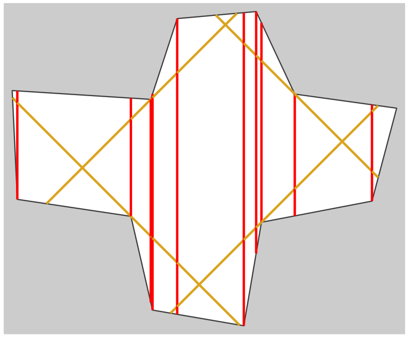

Fig. 6. Refinement of the decomposition from Figure 5 by vertical ray extensions upward and downward from each vertex. The resulting cells are convex.

C. Pursuit-Evasion Graph

In this section, we describe our algorithm which utilizes the convex conservative decomposition to construct its primary data structure, called the **pursuit-evasion graph (PEG)**. This section describes the vertices and edges the PEG.

The basic idea is that each node of the PEG corresponds to one element of the convex conservative decomposition including interior faces and the edges and vertices that surround each face, with a few important exceptions.

- The unbounded face of the DCEL, which represents the obstacle region $\mathbb{R}^2 - W$, does not generate any PEG nodes.
- The halfedges and vertices created during the decomposition do not generate PEG nodes, because many of these correspond to positions at which the gaps are changing. For clarity, we instead include PEG edges that transition directly between the associated faces (resp. edges) without stopping at the dividing half-edge (resp. vertex).

All other elements of the DCEL generate PEG nodes. This detail is important, because without coming into contact with the boundary of W, the pursuer can never clear any gaps.

Along with a specific convex conservative region, each PEG node is also associated with a unique set of clear/contaminated labels for each of the gaps that exist in that region. (Note that, because these regions are conservative, the set of gaps is the same for every point within a given region.) Thus, each bounded face in the DCEL generates 2^m nodes in the PEG; depending on the directions of the beams and the environment boundary, there will be between 2 and 2^{m+1} PEG nodes associated with each edge and vertex represented in the graph.

To compute the edges of the PEG, we iterate over each PEG node (which is already associated with gap labels) and consider each of its neighbors in the DCEL. It remains only to determine which PEG node for that region the directed edge should connect to. We can categorize the gap update rules that occur when transitioning between a source PEG node and a target PEG node into one of five cases, based on the dimension—that is, face (F), edge (E), or vertex (V)—of the source and target nodes.

a) $F \to F$ and $E \to E$ transitions: When the pursuer transitions from a face to an adjacent face (skipping a ray extension in the interior of W) or from an edge to an adjacent edge (skipping the endpoint of such a ray extension), we must update the clear/contaminated labels for the appropriate gaps. These transitions occur at the Left and Right events described in Section IV-B. There are four different event types lead to two different kinds of update rules.

1) At a Left-Extend event, the evader can hide behind the obstacle touched by b_i until after the beam has passed, and then contaminate the gap to the left of b_i. At a Right-Retract event the same effect occurs in reverse. Thus when the pursuer passes a Left-Extend or Right-Retract event from beam b_i, we assign:

$$g_{i-1} \leftarrow (g_{i-1} \text{ or } g_i).$$

All other gaps retain the same labels.

2) For Left-Retract and Right-Extend events, the cross contamination occurs in the opposite direction, so instead we assign:

$$g_i \leftarrow (g_i \text{ or } g_{i-1}).$$

Again, the other gaps do not change at this transition, so their gap labels remain unchanged.

Figures 7 and 8 illustrate these update rules.

b) $F \to E$ and $E \to F$ transitions: A second class of transitions moves from the interior of W to a boundary edge, or back again. Such movements can change the set of gaps in a variety of ways:

- A gap can appear or disappear (when both of its incident beams are aimed directly into the wall).
- A gap can shrink or grow (when only part of the gap is pressed against the wall).
- A gap can split into multiple gaps (when a wide gap is separated into two parts by coming into contact with the environment boundary).
- Multiple gaps can merge into a single gap (when a gap is re-joined with itself after leaving the environment boundary).

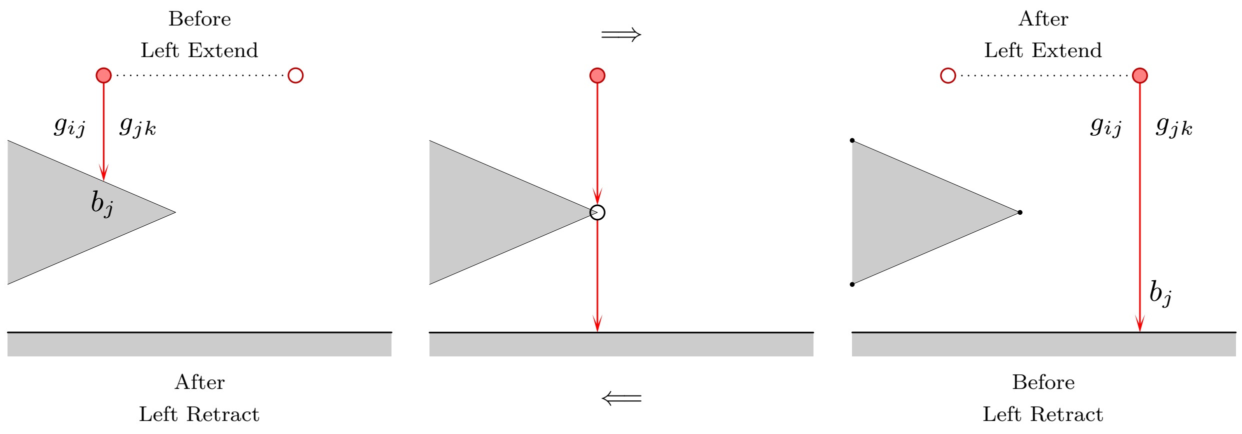

Fig. 7. An illustration of the **Left** Extend/Retract events.

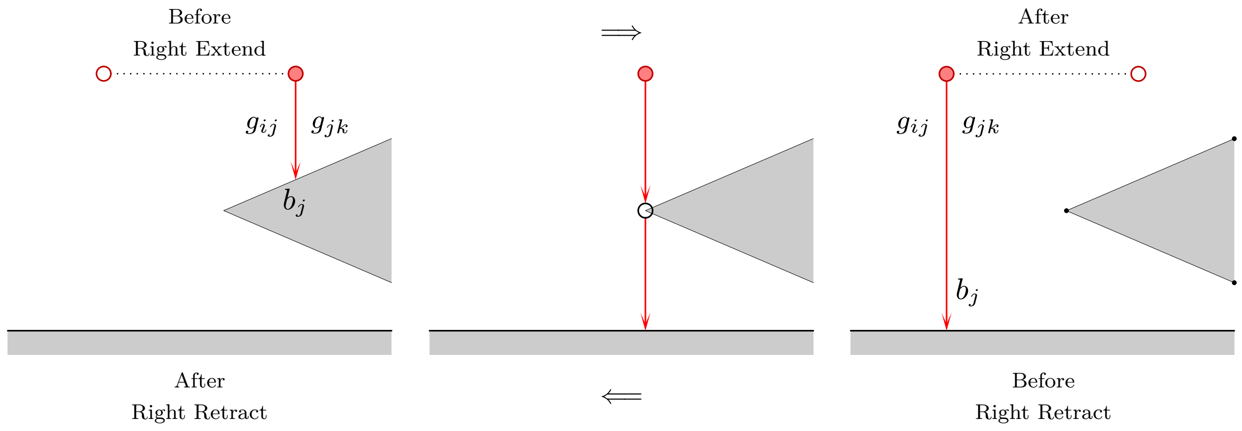

Fig. 8. An illustration of the **Right** Extend/Retract events.

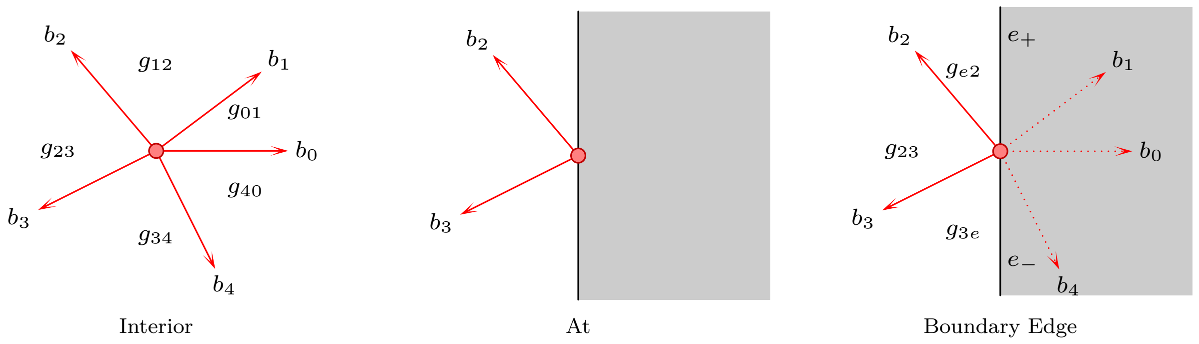

Fig. 9. A scenario where gap edges can appear/disappear and shrink/grow. When travelling from the interior to the boundary edge g_{01} and g_{40} disappear, g_{12} and g_{34} shrink to g_{e2} and g_{3e}, and g_{23} remains the same. Conversely, when travelling from the boundary edge to the interior gap edges appear, grow, and remain the same.

To handle all of these cases in a clean and compact way, we use a series of **gap containment** tests that determine whether the interior of one gap overlaps the interior of another. Such tests can be performed in constant time using a series of cw and ccw tests on the beam vectors. Then each gap in the target PEG node is marked as contaminated if and only if it overlaps at least on contaminated gap for the source node. Figure 9 illustrates one of these transitions.

c) $V \to E$ and $E \to V$ transitions: Transitions from an edge to a vertex or from a vertex to an edge can be handled identically to the $F \to E$ and $E \to F$ cases, with one important exception: If the vertex is a reflex vertex, then it is possible that some beams will emerge from (or disappear into) the environment instantaneously (rather than the gradual appear/disappear changes that occur for F/E transitions).

To handle this case properly, we must introduce an intermediate step, in which the gaps are computed at the reflex vertex, but only for beams which do not extend into the environment boundary in *both* the source and target regions. Figure 10 illustrates one of these transitions, for which the intermediate step is computed at the environment vertex. We use a series of gap overlap tests

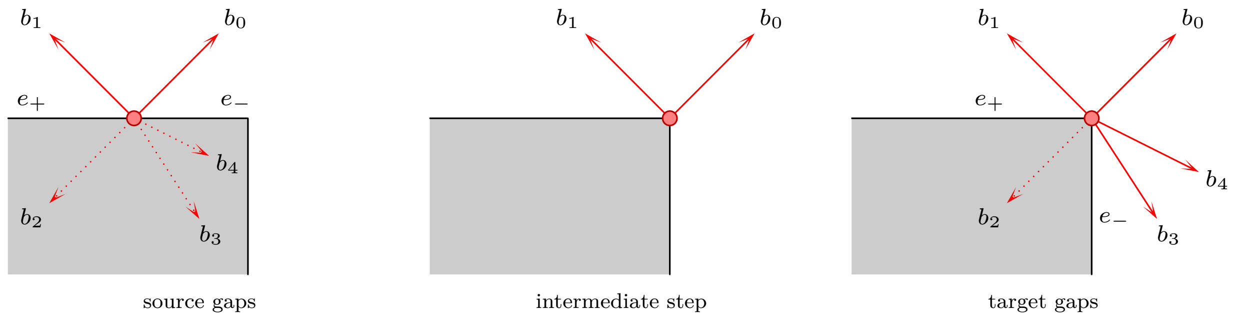

Fig. 10. An illustration of the **split** and **merge** events that occur when transitioning between an edge of the region graph and an environment vertex.

to propagate contamination forward from the source PEG node, correctly allowing the move past beams that are blocked by the edge (whether it is the source or target node). We then use the clear/contaminated labels from these intermediate gaps to populate the labels in the target node.

d) $F \to V$ and $V \to F$ transitions: Our algorithm omits direct transitions between faces and vertices for simplicity. Particularly for the case of reflex vertices, the correct assignment of gap labels is non trivial, because the algorithm must correctly identify which events to apply. This omission does not impact the correctness nor completeness of the algorithm, because any solution that traverses directly between a vertex and face can achieve the same result indirectly via the corresponding edge. (This does, of course, potentially make some of the final paths slightly longer.)

e) $V \to V$ transitions: The final of the nine cases is $V \to V$, which cannot occur because vertices are never adjacent in a DCEL.

Taken together, this set of nodes and edges fully captures the possibilities for the evader's location as a function of the pursuer's movements across the conservative regions.

D. Path Generation

The final step of the algorithm is a forward search through the PEG. The search starts from the pursuer's initial position with all of the gaps labeled as contaminated, and terminates when it reaches a PEG node in which all of the gaps are labeled clear. We use breadth-first search, though any graph search would be suitable.

Given a path from all-contaminated to all-clear, we generate the pursuer's final path in the usual way for cell-decomposition-based planning: We chain together the centroids of each region visited along the PEG path. The only complication is that, for $F \to F$ transitions, we must also include the midpoint of the edge separating those faces. The resulting path is guaranteed to locate the evader. Since the regions are all convex, the resulting path is guaranteed to stay within W, and to visit the PEG nodes in the correct order. Figure 11 completes the running example from Figures 5 and 6, illustrating a plan that correctly locates the evader.

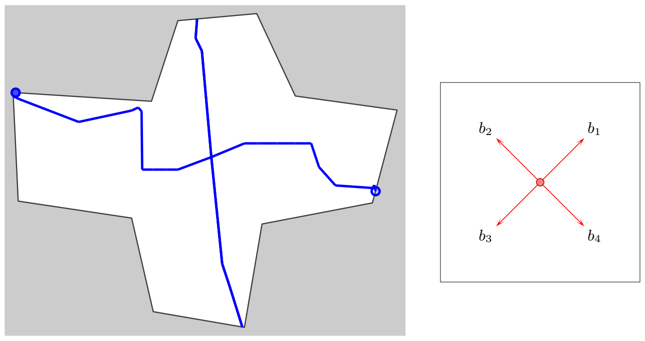

Fig. 11. The final generated plan for the example shown in Figures 5 and 6.

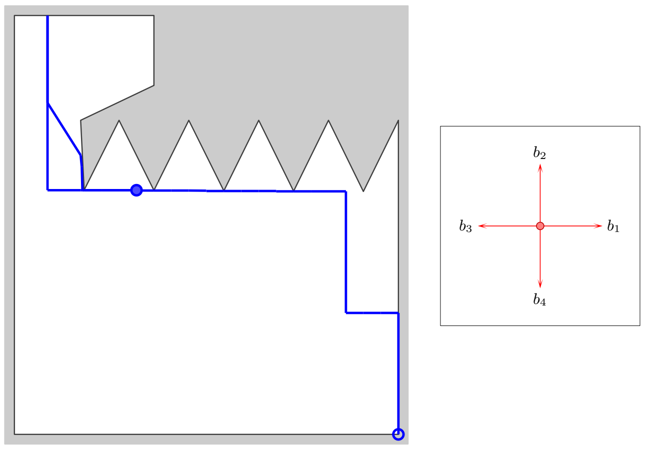

Fig. 12. A plan generated by our algorithm for the above environment.

Finally, because we know that the sequence of conservative regions is sufficient to characterize a solution, we know that if the PEG does not contain a path from the start node to an all-clear node, then the underlying pursuit-evasion problem has no solution.

E. Runtime analysis

The run time of this algorithm is dominated by the time needed to search the PEG, which has $O(2^m n^2)$ nodes and $O(2^m n^2)$ total edges. Therefore, the algorithm takes time $O(|V| + |E|) = O(2^m n^2)$.

V. Simulation Results

This section presents some example pursuit strategies computed by our implementation of this algorithm. Three examples appear in Figure 1, 11, and 12. With this implementation, which uses C++, a machine utilizing a single core of an Intel i5 processor and running the Gnu/Linux operating system was able to solve each of these instances in less than 0.1 seconds.

VI. Conclusion

In this paper the authors' present a complete algorithm for solving a pursuit-evasion problem in a simply-connected two-dimensional environment, for the case of a single pursuer equipped with fixed beam sensors. The algorithm constructs a DCEL by decomposing the environment based on critical gap events and further refines the partition by employing a trapezoidal decomposition to ensure a convex conservative decomposition. The decomposition induces a PEG, which is exhaustively searched and returns either a path through the PEG which corresponds to a pursuer motion strategy through the environment which is guaranteed to capture an evader, or reports failure for the current beam configuration.

Acknowledgement

We acknowledge support for this work from NSF (IIS-0953503).

References

[1] S. Bhattacharya and S. Hutchinson. A cell decomposition approach to visibility-based pursuit evasion among obstacles. *International Journal of Robotics Research*, 30(14):1709–1727, 2011.

[2] W. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.

[3] J. W. Durham, A. Franchi, and F. Bullo. Distributed pursuit-evasion without mapping or global localization via local frontiers. *Autonomous Robots*, 32(1):81–95, 2012.

[4] B. P. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research*, 25(4):299–315, 2006.

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

[6] Y. C. Ho, A. Bryson, and S. Baron. Differential games and optimal pursuit-evasion strategies. *IEEE Transactions on Automatic Control*, 10(4):385–389, October 1965.

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

[8] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion in a polygonal environment. *IEEE Transactions on Robotics*, 5(21):864–875, 2005.

[9] T. Kameda, M. Yamashita, and I. Suzuki. On-line polygon search by a seven-state boundary 1-searcher. *IEEE Transactions on Robotics*, 22(3):446–460, June 2006.

[10] K. Klein and S. Suri. Capture bounds for visibility-based pursuit evasion. In *Proc. ACM Symposium on Computational Geometry*, pages 329–338, 2013.

[11] A. Kleiner and A. Kolling. Guaranteed search with large teams of unmanned aerial vehicles. In *Proc. IEEE International Conference on Robotics and Automation*, 2013.

[12] A. Kolling and S. Carpin. Surveillance strategies for target detection with sweep lines. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5821–5827, 2009.

[13] A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 26(1):32–47, 2010.

[14] S. M. LaValle, H. H. González-Baños, C. Becker, and J.-C. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proc. IEEE International Conference on Robotics and Automation*, pages 731–736, 1997.

[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 on Computational Geometry and Applications*, 12(1-2):87–113, 2002.

[17] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[18] R. Murrieta, A. Sarmiento, S. Bhattacharya, and S. A. Hutchinson. Maintaining visibility of a moving target at a fixed distance: The case of observer bounded speed. In *Proc. IEEE International Conference on Robotics and Automation*, 2004.

[19] R. Murrieta-Cid, R. Monroy, S. Hutchinson, and J-P Laumond. A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *Proc. IEEE International Conference on Robotics and Automation*, 2008.

[20] N. Noori and V. Isler. Lion and man with visibility in monotone polygons. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, volume 86 of *Springer Tracts in Advanced Robotics*, pages 263–278. Springer, 2013.

[21] J. M. O'Kane and S. M. LaValle. On comparing the power of robots. *International Journal of Robotics Research*, 27(1):5–23, January 2008.

[22] S. Park, J. Lee, and K. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searcher. In *Proc. International Colloquium on Automata, Languages and Programming*, pages 281–290. Springer-Verlag, 2001.

[23] 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.

[24] N. N. Petrov. A problem of pursuit in the absence of information on the pursued. *Differentsial'nye Uraveniya (Differential Equations)*, 18:1345–1352, 1982.

[25] N. N. Petrov. The cossack-robber differential game. *Differentsial'nye Uraveniya (Differential Equations)*, 19:1366–1374, 1983.

[26] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[27] S. Rajko and S. M. LaValle. A pursuit-evasion bug algorithm. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1954–1960, 2001.

[28] U. Ruiz and R. Murrieta-Cid. Time-optimal motion strategies for capturing an omnidirectional evader using a differential drive robot. *IEEE Transactions on Robotics*, 21(3), June 2013.

[29] S. Sachs, S. M. LaValle, and S. Rajko. Visibility-based pursuit-evasion in an unknown planar environment. *International Journal of Robotics Research*, 23(1):3–26, 2004.

[30] J. Sgall. Solution of David Gale's lion and man problem. *Theor. Comput. Sci.*, 259(1-2):663–670, 2001.

[31] N. M. Stiffler and J. M. O'Kane. Shortest paths for visibility-based pursuit-evasion. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3997–4002, 2012.

[32] N. M. Stiffler and J. M. O'Kane. A complete algorithm for visibility-based pursuit-evasion with multiple pursuers. In *Proc. IEEE International Conference on Robotics and Automation*, 2014.

[33] N. M. Stiffler and J. M. O'Kane. A sampling based algorithm for multi-robot visibility-based pursuit-evasion. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, page To appear, 2014.

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

[35] B. Tovar and S. M. LaValle. Visibility-based pursuit-evasion with bounded speed. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2006.