Mapping and Pursuit-Evasion Strategies For a Simple Wall-Following Robot

Max Katsev, Anna Yershova, Benjamín Tovar, Robert Ghrist, and Steven M. LaValle

***Abstract*—This paper defines and analyzes a simple robot with local sensors that moves in an unknown polygonal environment. The robot can execute wall-following motions and can traverse the interior of the environment only when following parallel to an edge. The robot has no global sensors that would allow precise mapping or localization. Special information spaces are introduced for this particular model. Using these, strategies are presented for solving several tasks: 1) counting vertices, 2) computing the path winding number, 3) learning a combinatorial map, called the *cut ordering*, that encodes partial geometric information, and 4) solving pursuit-evasion problems.**

***Index Terms*—Exploration, information spaces, minimal sensing, pursuit-evasion, SLAM.**

I. Introduction

Imagine designing motion strategies for a simple, low-cost, differential-drive robot. The main objective in this paper is to investigate what kinds of global information can be learned and what kinds of tasks can be accomplished with as little sensing and actuation as possible. In a planar, indoor environment, wall-following is a simple operation that is easily accomplished using a contact sensor or short-range infrared sensor. Suppose the walls are polygonal and the robot approaches a vertex. If the interior angle at the vertex is greater than π, then it is possible for the robot to move past the wall by continuing to travel in the direction that the wheels are pointing. This case is called a *reflex vertex*. See Figure 1(a). These assumptions lead to a motion model that allows following walls and occasionally extending beyond the wall until another wall is contacted. Suppose that sensors can be used to determine whether the robot is at a reflex vertex, a convex vertex (interior angle less than π), the interior of an edge, or the interior of the environment. This is shown in Figure 1(b). The robot has no sensors that can measure precise distances or angles.

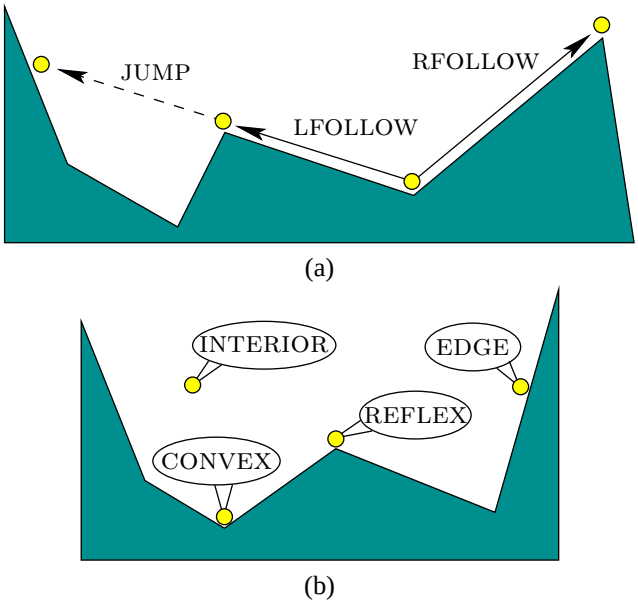


Fig. 1. (a) The robot can execute three movements: following the wall when it is to the right (RFOLLOW), following it on the left (LFOLLOW), and jumping straight to the next wall, traveling past a reflex vertex (JUMP). (b) The robot has a sensor that can distinguish between being in the interior, at a convex vertex, at a convex vertex, or in the interior of an edge.

Such a motion model is realistic for many low-cost, widely available platforms, such as iRobot Roomba or Lego NXT. To demonstrate the model, we implemented some of the motion primitives using Lego NXT platform and several proximity sensors. We performed several experiments in polygonal environments constructed from cinder blocks, showing that the jump motion of Figure 1(a) can be reliably executed. This allowed the robot to perform some of the tasks described in Sections IV and V (see Figure 2)[1]. Of course, these motions could be implemented in many other ways; the main focus of this paper is to understand what can be theoretically accomplished with such a cheap robot.

What kind of tasks can be accomplished with such a simple model, when the robot is dropped into an unknown environment? This question is answered from Sections IV to VI, which represent the main technical contributions of this paper. Before these are presented, related literature and basic definitions are provided in Sections II and III, respectively. Following this, Section IV shows that the robot can accomplish two simple tasks: counting the number of vertices and deter-

Manuscript received February 18, 2010; revised September 5, 2010. This work was supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

Max Katsev (corresponding author) is with the Department of Computer Science, University of Illinois, Urbana, IL 61801 USA (email: katsev1@uiuc.edu).

Anna Yershova is with the Department of Computer Science, Duke University, Durham, NC 27707, USA (e-mail: yershova@cs.duke.edu).

Benjamín Tovar is with the Department of Mechanical Engineering, Northwestern University, Evanston, IL 60208 USA (email: b-tovar@northwestern.edu).

Robert Ghrist is with the Departments of Mathematics and Electrical/Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA (email: ghrist@seas.upenn.edu).

Steven M. LaValle is with the Department of Computer Science, University of Illinois, Urbana, IL 61801 USA (email: lavalle@uiuc.edu).

[1]AVI format movie clip demonstrating the robot performing these tasks is available at http://ieeexplore.ieee.org

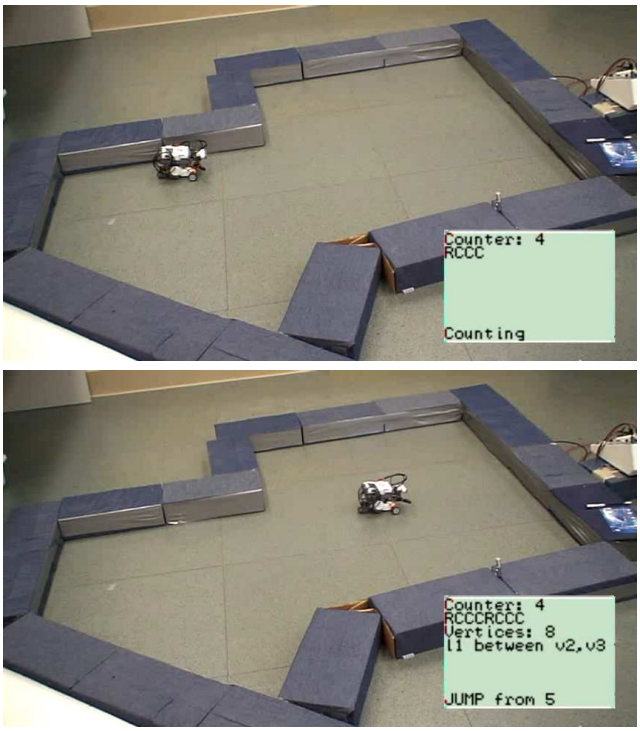


Fig. 2. Two frames of a video that shows the robot moving along a polygonal wall and "jumping" from a reflex vertex.

mining the number of times the robot "wrapped around" the boundary. It is furthermore established that a *pebble* (common in on-line exploration [2], [6], [13]) is required to accomplish these tasks.

Section V considers a combinatorial mapping and localization problem. The *cut ordering* is introduced, which is a new map that encodes precisely the geometric information that can be learned using the simple robot. We introduce a strategy that learns the cut ordering using a quadratic number of robot motions in terms of the number of polygonal environment edges. By building on the cut ordering, Section VI considers the pursuit-evasion problem, which involves systematically searching for an unpredictable moving target in the environment. This problem is considerably difficult because the environment is unknown, the robot cannot learn its precise structure, and it must pin down an elusive moving target. We introduce complete strategies for models that equip the robot with moderately more powerful sensors, which still cannot measure distances or angles. Completeness means that if a solution exists, the algorithm must find it; otherwise, it must report failure. We also introduce a strategy for the case of the weakest sensors; however, it may or may not be complete. It is based on conservatively approximating the pursuit status at every step, which leads to strategies that are guaranteed to find any and all unpredictable evaders. Finally, Section VII describes numerous interesting questions and open problems that are based on models considered in this paper. Parts of this work appeared in preliminary form in [65].

II. Related Work

At the highest level, there are numerous efforts in robotics literature that attempt to accomplish tasks with as little sensing as possible. Examples include sensorless manipulation [12], [16], [18], [21], [41], bug strategies [27], [28], [35], [42], and gap navigation trees: [33], [45], [62]. On-line exploration strategies make simple motion models and try to reduce the amount of memory or total distance traveled [5], [10], [19], [20], [29], [30], [43], [50], [56].

Most of these works that aim at understanding minimal requirements involve defining and analyzing *information spaces* associated with the sensing and actuation models (see [36], Chapter 11). The general idea is that the space of sensing and actuation histories can be compressed into smaller spaces that are used for filtering and planning, without requiring full state estimation. The basic concept of an information space can be traced back to work of Kuhn [31] in the context of game trees. There, the nondeterministic information state is referred to as an *information set*. After spreading throughout game theory, the concept was also borrowed into stochastic control theory (see [3], [32]). The term *information state*, I-state for short, is used extensively in [1] in the context of sequential and differential game theory. For further reading on information spaces in game theory, see [1], [49]. In artificial intelligence literature, I-states are referred to as *belief states* and are particularly important in the study of Partially Observable Markov Decision Processes (POMDPs). In robotics literature, they have been called *hyperstates* [22] and *knowledge states* [17]. Concepts closely related to I-spaces also appear as *perceptual equivalence classes* in [14] and the *information invariants* in [13].

There are numerous related works on localization, mapping, or both, often referred to as SLAM (Simultaneous Localization and Mapping). Most of this work focuses on exploring an information space that represents probability distributions over all possible configurations and environments [7], [11], [52], [61], [64]. Aside from our previous paper [65], the most related work is [59], in which the *combinatorial visibility vector* (cvv) is introduced as a sensing models that allows a minimalist robot to count the number of holes in an unknown polygonal environment. The model indicates the numbers of environment vertices that are visible between each depth discontinuity when performing an angular sweep. The information is combinatorial; however, the sensing range is unbounded. In Sections IV and V, we will consider localization and mapping problems using sensors that have only local range (for example, contact sensors).

Although mapping and localization is an important, basic operation, we often want robots to solve more complex tasks, such as tracking or searching for moving targets. Section VI addresses a pursuit-evasion problem that involves finding an unpredictable moving target in an unknown environment using our robot with weak sensing and motion capabilities. Pursuit-evasion problems in general were first studied in differential game theory [1], [25]. Pursuit-evasion in a graph was introduced in [53], and related theoretical analysis appears in [4], [34], [44]. Visibility-based pursuit-evasion was introduced in

[60], and the first complete algorithm appeared in [37]. An algorithm that runs in $O(n^2)$ for a single pursuer in a simple polygon was given in [51]. Variations that consider curved environments, beams of light, and other considerations appear in [8], [9], [15], [40], [58], [63]. Pursuit-evasion in three dimensions is discussed in [39]. Versions that involve minimal sensing and no prior given map are most closely related to Section VI: [24], [26], [55], [57], [65].

III. Basic Definitions

A. State, Action, and Observation Spaces

The robot is modeled as a point that can translate and rotate in a simply connected polygonal environment. The configuration space of the robot is $SE(2)$, in which each configuration is represented by (x_p, y_p, θ), with $(x_p, y_p) \in \mathbb{R}^2$ as the robot position and $\theta \in S^1$ as the orientation. It is assumed that the environment $E \subset \mathbb{R}^2$, an obstacle-free region, is the closure of a simply connected, bounded, polygonal open set. The environment is unknown to the robot; therefore, let $\mathcal{E}$ be the set of all possible environments. Let ∂E denote the boundary of $E \in \mathcal{E}$. Note that each $E \in \mathcal{E}$ can be encoded by specifying the vertices along ∂E in cyclic order. We make a general position assumption by restricting $\mathcal{E}$ only to include environments that contain no three collinear vertices.

In addition to the robot, the environment may contain a *pebble*, which is a special point that can be detected and moved by the robot. This will help the robot to recognize when it revisits a place. If the robot and pebble positions are identical, then the robot may or may not be carrying the pebble. Let $Q = \{0, 1\}$ represent the set of values for a state variable q, in which $q = 1$ means that the robot is holding the pebble; otherwise, $q = 0$.

Let X be the *state space*, which encodes all possible configurations for the robot and the pebble in the environment. Possible configurations of the robot are a subset of $SE(2)$, whereas for the pebble are a subset of $\mathbb{R}^2$. If E were given in advance, then a reasonable choice for the state space would be $X \subset SE(2) \times \mathbb{R}^2 \times Q$, which could be parametrized in particular as $X = E^2 \times S^1 \times Q$. For the problems in this paper, however, the environment is unknown and properties of it are discovered as the robot moves. Therefore, the state space is defined as:

$$X \subset SE(2) \times \mathbb{R}^2 \times Q \times \mathcal{E}. \tag{1}$$

For a particular state, we require that both the position of the robot and the pebble to be inside the environment.

The robot sensors are modeled as follows. Let Y be an *observation space*, which is a set of possible sensor readings. A *sensor mapping* $h : X \to Y$ is defined that indicates what the sensor is supposed to observe from state $x \in X$. Two sensors mappings are defined. For the first one, the *touch sensor*, consider the robot's position $(x_p, y_p) \in E$. Every environment E can be partitioned into four sets: 1) the interior of E, 2) the interior of an edge along ∂E, 3) a convex vertex (interior angle less than π), and 4) a reflex vertex (interior angle greater than π). The touch sensor $h_t : X \to Y_t$ yields an observation that correctly determines which of these four sets contains (x_p, y_p). The observation space is

$$Y_t = \{\text{INTERIOR}, \text{EDGE}, \text{CONVEX}, \text{REFLEX}\}. \tag{2}$$

The second sensor mapping, *pebble sensor*, considers the position of the robot and the pebble in E. The pebble sensor $h_q : X \to \{0, 1\}$ indicates with $h_q(x) = 1$ if the robot and pebble positions are identical; otherwise, $h_q(x) = 0$. These two sensors are combined into a single sensor mapping $h : X \to Y_t \times \{0, 1\}$, which yields $y = h(x)$ from any $x \in X$.

An *action space* U is defined to model robot motions. Each *action* $u \in U$ causes the robot to move until some internal termination condition is met. This results in a set of discrete *stages*, in which stage $i = 1$ is the initial stage, and stage $i = k$ is the resulting stage after $k - 1$ actions have been applied. A *state transition function* $f : X \times U \to X$ is defined, which yields a new state x_{k+1} when $u_k \in U$ is applied from some $x_k \in X$.

For the robot model in this paper, U is defined as the set of the following actions (the first three were shown in Figure 1(a)).

1) $u =$ RFOLLOW, which traverses an edge in the counterclockwise direction until either the next vertex or the pebble is reached. This action can only be applied when the robot is making contact with ∂E, and during execution, the edge transversed is to the right of the robot.
2) $u =$ LFOLLOW, which traverses an edge in the clockwise direction until a vertex or the pebble is reached. Analogously to RFOLLOW, the edge is to the left of the robot, and the robot is in contact with ∂E.
3) $u =$ JUMP, which is applicable only from a reflex vertex. Assume that the robot arrived at the reflex vertex after traversing a wall. When $u =$ JUMP is applied, the robot continues to move straight into the interior of E until ∂E is hit again.
4) $u =$ GRAB, which picks up the pebble, enabling the robot to carry it. This action can only be applied if the robot and pebble are at the same position.
5) $u =$ DROP, which places the pebble at the current robot position.
6) $u =$ INIT, which applies from any configuration and terminates whenever the robot reaches any vertex of ∂E. Imagine the robot uses a standard differential-drive mechanism. The robot can move straight from the interior of E until a wall is hit and then follow the wall in an arbitrary direction (say RFOLLOW) until a vertex is reached. Assume that once the vertex is reached, the wheels are pointing in the direction parallel to the wall that was just traversed.

B. Information Spaces

Although we assume that the state *space* is known, the particular *state* will be, in general, unknown to the robot. Therefore, we need to be precise about what information the robot has available. In general, such information is called an *information state* or *I-state* for short. For further details and alternative formulations of information spaces, see Chapter 11 of [36].

This most direct and basic I-state will be called the *history I-state*, and is defined at stage k as

$$\eta_k = (u_1, \ldots, u_{k-1}, y_1, \ldots, y_k), \tag{3}$$

which is simply the sequence (or "memory") of all actions taken and observations received up to stage k. The set of all possible η_k for all possible k is called the *history I-space* and is denoted by $\mathcal{I}_{hist}$.

Although $\mathcal{I}_{hist}$ is natural because it arises directly from the problem, it is difficult to analyze, due in part to the linear growth of I-state components with respect to k. This motivates the construction of mappings that attempt to project $\mathcal{I}_{hist}$ down to a "smaller" space that will be more manageable for analysis and computation. Let $\mathcal{I}_{der}$ be any set and consider a mapping $\kappa : \mathcal{I}_{hist} \to \mathcal{I}_{der}$. In general, $\mathcal{I}_{der}$ is called a *derived I-space* and κ is called an *information mapping* or *I-map*. Ideally, $\mathcal{I}_{der}$ and κ should be chosen so that an *information transition function* can be defined:

$$\kappa(\eta_{k+1}) = f_{der}(\kappa(\eta_k), u_k, y_{k+1}). \tag{4}$$

This means that $\kappa(\eta_k)$ can be computed incrementally without storing elements of $\mathcal{I}_{hist}$. The derived I-state $\kappa(\eta_k)$, which is usually smaller, can be used together with u_k and y_{k+1} to obtain η_{k+1}. An example of this occurs in the Kalman filter, in which the current mean, covariance, action, and observation are sufficient for obtaining the new mean and covariance, rather than referring back to the complete history I-state. In one trivial case, κ is the identity function, which yields

$$\eta_{k+1} = f_{hist}(\eta_k, u_k, y_{k+1}), \tag{5}$$

based on simply inserting u_k and y_{k+1} into η_k to obtain η_{k+1}.

If a mapping of the form in (4) exists, then a kind of filter can be made that essentially "lives" in $\mathcal{I}_{der}$, rather than $\mathcal{I}_{hist}$. The goal in the coming sections will be to choose $\mathcal{I}_{der}$ and κ carefully so that the derived I-space can be analyzed and the derived I-states contain information that is sufficient for solving a specified task.

IV. Counting Windings and Vertices

Now we consider basic filtering problems, which includes determining simple properties of the robot path and the environment. The concepts in this section are somewhat straightforward; however, they serve to illustrate the technical definitions and concepts of Section III, which are critical for later sections.

We consider a model in which only the actions INIT, RFOLLOW, and LFOLLOW are available, and that the pebble is fixed at some vertex. This brings a couple of restrictions. First, the robot can sense the pebble, but it cannot manipulate it. Second, it can move from vertex to vertex, but cannot jump and cannot determine whether a vertex is convex or reflex. To simplify the expressions below, assume that in an initial stage $i = 0$, $u_0 =$ INIT is successfully applied so that robot is already at a vertex of E.

A. Determining the Winding Number

The first task is to determine the number of times that the robot has *wrapped* around ∂E. This is called the *winding number*, and is the number of times the robot has traveled around ∂E by systematically eliminating all reversals. In a continuous setting, this is obtained by taking the shortest path within its homotopy class. The winding number can be positive, negative, or zero. A positive winding number means that the robot wrapped counterclockwise around ∂E, and negative means clockwise.

We now introduce derived I-spaces to compute interesting statistics based on the history I-state. For this, let $u_i \in U$ be the action applied at stage i. For u_i, let $a_i = 1$ if $u_i =$ LFOLLOW, $a_i = -1$ if $u_i =$ RFOLLOW, and $a_i = 0$ otherwise.

The I-map

$$\kappa_1(\eta_k) = \sum_{i=1}^{k-1} |a_i| \tag{6}$$

indicates the total number of edges traversed by the robot. The right side refers to a_i, which is derived from u_i, and is included in η_k, the argument to κ_1. Note that κ_1 can be implemented recursively as a filter:

$$\kappa_1(\eta_{k+1}) = \kappa_1(\eta_k) + |a_k|, \tag{7}$$

which is in the form of (4). Hence, it is possible to "live" in a derived I-space that indicates only the number of actions taken.

The I-map

$$\kappa_2(\eta_k) = \sum_{i=1}^{k-1} a_i \tag{8}$$

yields the distance traveled after eliminating all reversals. This is called the *combinatorial distance*, and is the number of edges in the shortest path among all those homotopic to the actual path taken by the robot, with the start and end points fixed.

If y_i is the observation at stage i, then let $w_i = 1$ if the pebble is detected, and $w_i = 0$ otherwise. The I-map

$$\kappa_3(\eta_k) = \sum_{i=1}^{k} w_i \tag{9}$$

yields the number of times the pebble has been contacted. Let $\kappa_4(\eta_k)$ be the smallest i for which $w_i = 1$.

Proposition 1 *The winding number at stage $k > \kappa_4(\eta_k)$ is given by*

$$\kappa_5(\eta_k) = \sum_{i=\kappa_4(\eta_k)+1}^{k-1} w_i(a_{i-1} + a_i)/2, \tag{10}$$

using the pebble location as the base point.

Proof: Consider a path that monotonically traverses ∂E counterclockwise m times, starting and stopping from a vertex other than the base point. The term $(a_{i-1} + a_i)/2$ yields 1 during the entire execution. Each time the pebble is crossed, $w_i = 1$. The pebble is crossed m times, and (10) therefore

yields the correct winding number. Now suppose that the monotonic path starts and stops at the pebble. The sum in (10) does not count the first pebble contact; however, the last pebble contact is counted once; hence, the correct winding number is obtained. By similar arguments, a clockwise monotonic path yields $-m$ because $(a_{i-1} + a_i)/2$ yields -1 each time the pebble is crossed.

Now consider non-monotonic paths. If a reversal occurs at the pebble, then $(a_{i-1} + a_i)/2$ yields 0, which is correct because the pebble was not crossed. If a path crosses the pebble counterclockwise and the next crossing is clockwise, then the corresponding two terms in (10) cancel, once again preserving the correct winding number. After all such cancellations occur, $\kappa_5(\eta_k)$ reports the correct winding number. ■

B. Counting Polygon Vertices

Now suppose that the robot needs to count the number of vertices that lie along ∂E. One possibility is to move counterclockwise until the pebble is encountered twice and make an I-map that subtracts the stage indices at which the pebble is contacted. To make the problem more interesting, consider how to make an I-map that does not constrain the robot to a particular path but allows it to nevertheless infer the number of vertices. In this case, a kind of passive filter is obtained for obtaining the vertex count.

As an intermediate step, define

$$\kappa_6(\eta_k) = \sum_{i=\kappa_4(\eta_k)+1}^{k-1} a_i, \tag{11}$$

which indicates the combinatorial distance relative to the first encounter of the pebble. Let $\kappa_7(\eta_k)$ be the minimum $i \leq k$ such that $\kappa_5(\eta_i)\kappa_6(\eta_i) \neq 0$, or 0 if there is no such i.

Proposition 2 *Let $i = \kappa_7(\eta_k)$, and $\kappa_8(\eta_k) = |\kappa_6(\eta_i)|$. If $\kappa_8(\eta_k) \neq 0$, then $\kappa_8(\eta_k)$ is the number of vertices in ∂E.*

Proof: If $\kappa_8(\eta_k)$ is zero, then either the combinatorial distance $\kappa_6(\eta_k)$ from the first encounter of the pebble is zero, or the winding number $\kappa_5(\eta_k)$ is zero. The first time $\kappa_5(\eta_k)\kappa_6(\eta_k)$ is different from zero occurs when the robot encounter the pebble after winding around ∂E exactly once, and the result follows. ■

C. Termination Issues

A pebble was used in the models above because the robot cannot solve the tasks without it (assuming the rest of the model remains fixed), as established by the following proposition:

Observation 3 *Without a pebble, it is impossible to compute the winding number or count the number of environment vertices.*

Proof: Consider an infinite sequence of regular polygons in which the number of vertices n increases incrementally from $n = 3$. Imagine that we place the robot in one of the regular polygons, without indicating which one it is. The robot is capable of taking counterclockwise or clockwise steps along ∂E, but it has no additional information that it can use to infer which polygon it is traveling in. Hence, it cannot count the number of vertices or the winding number if presented with this sequence of possible environments. Since this sequence is a strict subset of $\mathcal{E}$, it is not possible for the robot to compute the winding number or count the number of environment vertices. ■

V. Learning the Environment Structure

This section considers what can be learned about the environment using the actuation and sensing model defined in Section III. We now use the complete set of actions, the touch sensor, and the pebble sensor. We introduce a new combinatorial map, called the *cut ordering*, which precisely characterizes what can be learned about the environment and how the robot can localize itself combinatorially.

A. The Cut Ordering

Consider the paths traversed by the JUMP action from Section III and Figure 1(a). Each path can be viewed as a directed segment that starts at a reflex vertex and ends at a point on ∂E. Each such segment will be referred to as a *cut*. If the robot is following the wall to the left (the LFOLLOW action) before JUMP is applied, then it is called a *left cut*. Suppose that the vertices along ∂E are enumerated from v_1 to v_n in counterclockwise order. For a reflex vertex v_i, the terminal point of the left cut on ∂E is denoted as ℓ_i and is called the *cut endpoint*. Similarly, if the robot is following the wall to the right and jumps, then a *right cut* is obtained. The cut endpoint is denoted as r_i. See Figure 3 for a simple example. Note that every cut endpoint is *visible* from its associated reflex vertex. Two points in E are said to be (mutually) visible if the line segment that joins them is completely contained in E.

The set of all cuts of E together with ∂E, is called *the cut arrangement of* E. The combinatorial structure of a cut arrangement is determined by the order in which the cuts intersect in the interior of E, and by the order in which the endpoints of the cuts appear in ∂E.

The general position assumption introduced in Section III-A guarantees that no cut endpoint lands on another vertex. At this point, to simplify further presentation we also assume that no two cut endpoints land on each other. This implies that a point in ∂E cannot be collinear with two or more edges, if a cut emanates from each of the edges.

Let M be the complete collection of all vertices and all endpoints of cuts from reflex vertices. If an environment boundary has n vertices, $m < n$ of which are reflex, then M contains $n + 2m$ points. The *cut ordering* of an environment E is the cyclic permutation of M that is consistent with the ordering of all points in M as they appear along ∂E in

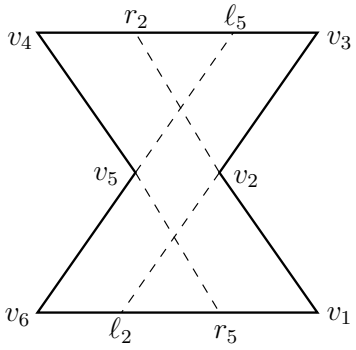


Fig. 3. An environment that has two reflex vertices and four associated cuts.

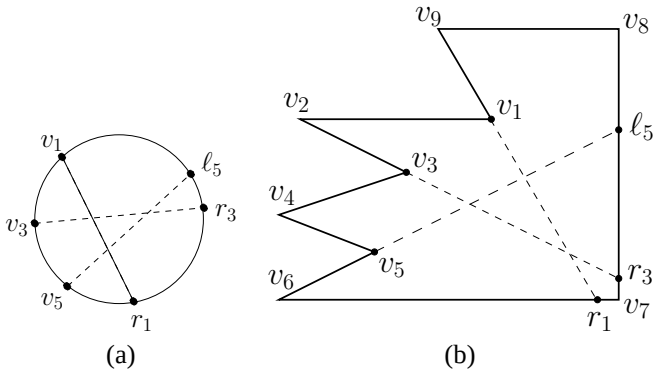


Fig. 4. The cut diagram indicates which cuts intersect, but it does not preserve the combinatorial structure of the cut arrangement. The segments in (a), which is a partial cut diagram, intersect differently from the actual cuts in E, which is shown in (b); note that in (a), $\overline{pr_p}$ passes to the left of the intersection of the other two segments; however, in (b), the corresponding cut passed to the left.

counterclockwise order. For the example in Figure 3, the cut ordering is

$$(v_1, v_2, v_3, \ell_5, r_2, v_4, v_5, v_6, \ell_2, r_5). \tag{12}$$

Since the ordering is cyclic, it can be equivalently expressed by starting from any element of M. Furthermore, the vertex numbering over ∂E is arbitrary. Assuming that vertices are named consecutively in counterclockwise order from v_1 to v_n, there are n possible ways to name vertices depending on which vertex is called v_1. Two cut orderings are said to be *equivalent* if the cyclic ordering is preserved after relabeling the vertices. For example, if in Figure 3 we relabel v_3 to be v_1 and enumerate the other vertices in counterclockwise order, then (12) becomes

$$(v_1, \ell_3, r_6, v_2, v_3, v_4, \ell_6, r_3, v_5, v_6). \tag{13}$$

This can be made more similar in appearance to (12) by cyclically shifting each index by two to obtain:

$$(v_5, v_6, v_1, \ell_3, r_6, v_2, v_3, v_4, \ell_6, r_3). \tag{14}$$

If two cut orderings are not equivalent, they are called *distinct*.

The cut ordering can be visualized geometrically by defining a *cut diagram* as shown in Figure 4(a) for the polygon in Figure 4(b). Take the points in M and point them around a circle in their proper cyclic order. Connect each reflex vertex v_i with a line segment to each of ℓ_i and r_i. This clearly identifies some points along ∂E that are mutually visible. The cut diagram is closely related to other structures for encoding geometric information in polygons, such as the visibility graph [46], [47], the chord diagram [60], the visibility obstruction diagram [38], and the link diagram [15].

Note that the cut diagram indicates segment crossing information from the original polygon, even though it is constructed entirely from the cut ordering:

Proposition 4 *For any environment E, each pair of cuts intersects if and only if their corresponding segments intersect in the cut diagram.*

Proof: Consider any pair of segments, $\overline{ss'}$ and $\overline{tt'}$, with distinct endpoints $\{s, s', t, t'\} \in \partial E$. They intersect in the interior of E if and only if the cyclic ordering of the endpoints along ∂E alternates between s or s' and t or t'. Examples are (s, t, s', t') and (s, t', s', t). If the cyclic ordering obtained by traveling around ∂E is (s, s', t, t'), for example, then the segments do not intersect. Likewise, the intersections of pairs of segments in the cut diagram are completely determined from the cyclic ordering of endpoints around the circle. Since the cut diagram preserves the cyclic ordering of endpoints along ∂E, the cuts intersect in E if and only if they intersect in the cut diagram. ■

Note however, that the cut diagram does not recover the full combinatorial structure of the cut arrangement of E. In other words, the cell decomposition induced by the cut arrangement does not necessarily correspond to the cell decomposition induced by the cut diagram. An example is shown in Figure 4.

B. Derived I-Spaces

Suppose that some actions have been executed by the robot and some sensor observations have been obtained. After k stages, this results in a history I-state η_k, as given in (3). To construct derived I-spaces, recall the state space $X \subset SE(2) \times \mathbb{R}^2 \times P \times \mathcal{E}$. Based on η_k, we would like to reason about the set of possible current states $x_k \in X$. It turns out that the cut ordering provides a convenient way to characterize these sets.

Recall the collection $\mathcal{E}$ of all environments, as defined in Section III. Every $E \in \mathcal{E}$ has a unique associated cut ordering, once the equivalence described in Section V-A is taken into account. Let C denote the set of all possible distinct cut orderings, for any $n \geq 3$ vertices and $m < n$ reflex vertices. Since each $E \in \mathcal{E}$ maps to a cut ordering, it is natural to ask whether the mapping from $\mathcal{E}$ to C is onto. This is not the case, as many cut orderings are not realizable. For example, let a *reflex chain* refer to a sequence of consecutive reflex vertices along ∂E. By simple geometry, it is clear that the cut endpoint of a vertex v along a reflex chain cannot appear between the vertices of the same chain. The edges incident to v block the cuts.

Note that for our problem, numerous environments have the same cut ordering. The preimages of the mapping from $\mathcal{E}$ to C

partition $\mathcal{E}$ into equivalence classes of polygonal environments that produce the same cut ordering. Polygons within a class may have quite different scales, relative edge lengths, and angles between edges.

Let the power set of C be denoted as $\mathcal{I}_{co}$, which is a derived I-space under an I-map $\kappa_{co} : \mathcal{I}_{hist} \rightarrow \mathcal{I}_{co}$. To define κ_{co}, let $\kappa_{co}(\eta_k)$ be the set of all cut orderings that are consistent with all of the data in η_k. As will be seen shortly, the cut ordering is incrementally constructed from η_k by moving the robot according to a specified plan. At any given time, a *partial* cut ordering has been learned. The set of all cut orderings into which the partial cut ordering can be embedded forms $\kappa_{co}(\eta_k)$. Intuitively, $\kappa_{co}(\eta_k)$ corresponds to all full cut orderings that could possibly be obtained by extending the current, partial cut ordering.

Therefore, a kind of localization and mapping problem arises. The problem is to construct a sequence of actions (or plan) that always results in a unique cut ordering, regardless of the particular initial configuration or environment $E \in \mathcal{E}$. Expressed differently, the goal is to obtain $|\kappa_{co}(\eta_k)| = 1$ after some number k of stages (the particular k may depend on the initial state).

C. Learning the Cut Ordering

Consider the following strategy[2]:

Strategy 1 *Learning the cut ordering*

Description: Initially, the robot executes INIT, drops a pebble using DROP, and executes a sequence of LFOLLOW actions until the pebble is reached again. As shown in Section IV, the number n of vertices can easily be counted. Furthermore, the touch sensor can be used to determine the location of each reflex vertex. Let the vertices be enumerated during execution, starting from 1 at the pebble, and let $F(E) \subset \{v_1, \ldots, v_n\}$ be the recorded set of reflex vertices of E.

To construct the cut ordering, the robot needs to determine where every left and right cut endpoint reaches ∂E. The precise location need not be determined; however, the cut ordering requires determining only the cyclic permutation of all vertices and cut endpoints. For each $v_i \in F(E)$, the robot must determine ℓ_i and r_i. The method proceeds inductively. To determine ℓ_i, the robot executes $u =$ LFOLLOW actions until vertex v_i is reached and then executes JUMP. After arriving on ∂E, the robot executes a sequence of m LFOLLOW actions until the pebble is reached. The robot infers that ℓ_i is between vertex v_{n-m+1} (mod$n$) and v_{n-m}(mod$n$). Similarly, the location of r_i is determined by a sequence of $u =$ RFOLLOW actions to reach vertex v_i, followed by a JUMP action, and finally a concluding sequence of m RFOLLOW actions to reach the pebble.

Based on the construction so far, the robot knows only the edges on which the cut endpoints lie; however, it does not know the ordering of the cut endpoints within an edge. To determine this ordering, a comparison operation can be executed for each pair of cuts that have endpoints on the same edge. For the first cut, its corresponding JUMP action is executed and a pebble is dropped using DROP at the cut endpoint. For the second cut, its corresponding JUMP action is executed. Following this, the robot executes RFOLLOW. If the pebble is encountered, then the first cut endpoint is to the right of the second one; otherwise, the order is reversed.

Proposition 5 *The robot can learn the cut ordering associated with E using $O(n^2)$ actions and $O(n)$ space, in which n is the number of vertices in ∂E.*

Proof: Using Strategy 1, the number of actions is bounded above by $O(n^2)$ since there are $O(n)$ actions executed for each cut and there are at most $O(n)$ cuts. There are exactly two cuts per reflex vertex; hence, the cut ordering and the strategy use $O(n)$ space. ■

This is clearly optimal in space, and it appears to be asympotically optimal in the number of actions because the robot has such weak sensors that it must traverse a linear number of edges to determine each cut endpoint.

The next proposition determines whether the pebble sensor is required for learning the cut ordering of E:

Observation 6 *Without sensing a pebble, the robot cannot construct the cut ordering.*

Proof: As in Observation 3, there exist polygons for which the robot cannot determine whether it has returned to a previous vertex. In the present setting, consider any convex polygon. There are no cuts and no additional information that can be used to recognize that the robot has returned to the initial vertex after winding around the polygon boundary. Hence, it cannot infer the number of vertices in ∂E, which is needed to construct the cut ordering. ■

It turns out that the cut ordering associated with E is the maximum amount of information that the robot can gather about reachable positions in the environment:

Proposition 7 *Once the cut ordering has been learned, no additional combinatorial information regarding the cut arrangement of E can be obtained.*

Proof: Consider the set of all possible action sequences, applied in some particular environment E, together with the points in E reached. After each action, the robot terminates at a particular point along ∂E. Let Z be the set of all possible positions along ∂E that can be reached by an action. The elements of Z correspond directly to vertices of E and all cut endpoints. Once the cut ordering has been learned, the cut ordering predicts precisely which point in Z will be reached by applying any action sequence from any initial position in Z. Thus, no "surprises" can be obtained by further exploration. The sensors are not powerful enough to

[2]We intentionally use the word *strategy* rather than *algorithm* to emphasize that the state (E, robot position, and pebble position) is unknown to the robot; therefore, it is not an *input* to an algorithm in the usual sense.

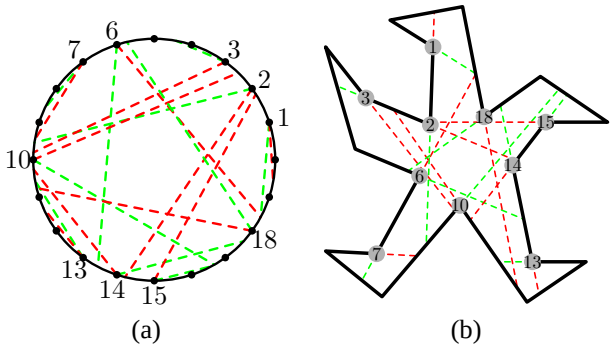


Fig. 5. For the given environment, the cut diagram with all of the cuts generated by our program, is shown on the left.

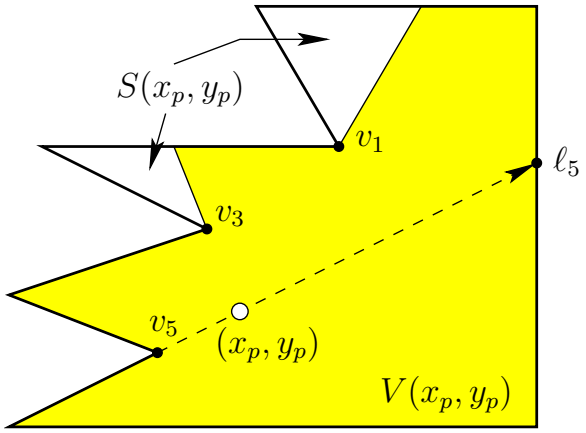


Fig. 6. When the robot is at some position (x_p, y_p), the detection sensor detects the evader if it lies in the visible region $V(x_p, y_p) \subseteq E$. The shadow region $S(x_p, y_p)$ is the complement, which corresponds to places where the evader cannot be detected from (x_p, y_p).

learn any information regarding precise distances; therefore, the ordering of points in Z along ∂E is the most that can be obtained. Therefore, the cut ordering corresponds to the maximal amount of combinatorial information about the cut arrangement of E. ■

The strategy was implemented in simulation, and a computed example is shown in Figure 5.

VI. Solving Pursuit-Evasion Problems

Now consider the challenging task of winning a pursuit-evasion game. The robot is a *pursuer* that must find one or more *evaders* that are initially hidden and move unpredictably through E. The robot has all of the sensors and actions defined in Section III.

A. Extending the Models

An additional sensor is needed to detect evaders. For now, assume there is only one evader. The coming approach will actually find *all* evaders if there are many; however, there is no need to complicate the notation at this stage. The evader is modeled as a point that starts at some unknown $(x_e, y_e) \in E$ and moves arbitrarily fast along a continuous, time-parametrized path that is unknown to the robot. The state space is extended from (1) to obtain

$$X \subset SE(2) \times \mathbb{R}^2 \times \mathbb{R}^2 \times P \times \mathcal{E}. \tag{15}$$

in which we included an additional $\mathbb{R}^2$ to represent $(x_e, y_e) \in E$. A *detection sensor*, $h_d : X \rightarrow \{0, 1\}$ yields $h_d(x) = 1$ if and only if the robot position (x_p, y_p) and the evader position (x_e, y_e) are mutually visible in the particular $E \in \mathcal{E}$. Note that the detection sensor provides no information about the structure of E; it yields only a single bit of information. The robot must rely on whatever information it can learn about E, which is precisely the cut ordering from Section V.

The task is to compute a sequence of actions, called a *plan*, that guarantees that the evader will be detected, regardless of the particular environment, the initial position of the robot (pursuer), the initial evader position, and the path taken by the evader.

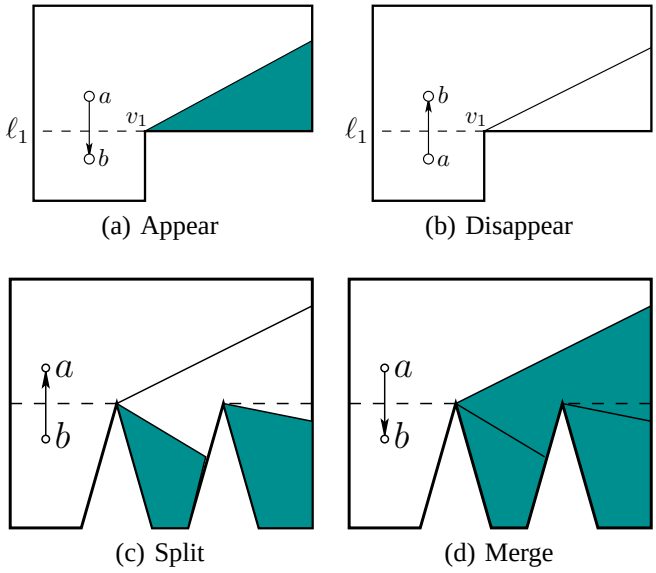
B. Solution Using a Gap Sensor

The planning problem is complicated by the challenge of maintaining the status of the pursuit as the robot moves. This corresponds to computing a derived I-state that indicates the set of states that are possible given the history I-state. This section gives the robot a sensor that enables it to exactly maintain the status and leads to a *complete* planning strategy. This means that the strategy computes a solution if one exists; otherwise, it reports failure after a finite number of steps. The given sensor is too powerful in this context; therefore, Sections VI-C through VI-E weaken the sensing requirement until the robot is left only with its binary detection sensor and the sensors of Section III.

Suppose the robot is at $(x_p, y_p) \in E$ and let $V(x_p, y_p) \subseteq E$ denote the *visibility region*, which is the set of all points visible from (x_p, y_p). The evader is detected if and only if $(x_e, y_e) \in V(x_p, y_p)$. Let the *shadow region* $S(x_p, y_p) = E \setminus V(x_p, y_p)$ be the set of positions where the evader is undetected. Figure 6 shows a simple example. One reasonable way to represent the pursuit status would be to maintain the set of possible hiding places for the evader. This means that $S(x_p, y_p)$ should be partitioned into two regions: 1) places where the evader *might* be, and 2) places where the evader *cannot* be.

Looking at Figure 6, it should be clear that if $S(x_p, y_p)$ is nonempty, then it must have a finite number of connected components, given that evader moves arbitrarily fast. Let these be called *shadow components*. Imagine placing a label of 1 on each shadow component that might contain the evader, and 0 on the remaining shadow components. This is sufficient for characterizing *any* pursuit status that might arise. For every shadow component, either all points are possible locations for the evader or none of them are. There is no need for multiple labels within a component. This observation forms the basis of the pursuit-evasion strategies in [23].

To proceed further, some terminology is needed. Traveling counterclockwise around ∂E, the right cut of a reflex vertex that is immediately preceded by a convex vertex is called a *right inflection*. The left cut of a reflex vertex that is



(a) Appear (b) Disappear

(c) Split (d) Merge

Fig. 7. The four types of events in terms of shadow components.

immediately followed by a convex vertex is called a *left inflection*; the dashed line in Figure 7(a) shows an example. Note that if both neighboring vertices of a reflex vertex along ∂E are convex, then both of its cuts are inflections.

Now we define the important notion of a bitangent. A line is *tangent* to a reflex vertex v if it contains v and both edges incident to v lie on the same side of the line. A *bitangent* is a maximal line segment contained in E, and whose supporting line is tangent at two distinct, mutually visible reflex vertices, say v_i and v_j. Since a bitangent is a maximal line segment, its endpoints are in ∂E. Let $b_{i,j} \in \partial E$ denote the endpoint of the bitangent that is closest to v_i. Likewise, let $b_{j,i} \in \partial E$ denote the endpoint closest to v_j. Any bitangent can be divided into three segments, connecting: 1) $b_{i,j}$ and v_i, 2) $b_{j,i}$ and v_j, and 3) v_i and v_j. The first two are called *bitangent rays*, and are illustrated by the dashed lines in Figure 7(c).

Now imagine having a powerful sensor that detects when a topological change occurs in $S(x_p, y_p)$. If the pursuer moves along a path, one of four topological *events* may occur in $S(x_p, y_p)$ (assuming general position for E):

1) **Appear:** A shadow component appears, which is caused by crossing an inflection as shown in Figure 7(a).
2) **Disappear:** A shadow component disappears, which is caused by crossing an inflection ray in the other direction; see Figure 7(b).
3) **Split:** A shadow component separates into two, which is caused by crossing a bitangent ray, which is shown in Figure 7(c).
4) **Merge:** Two shadow components merge into one, which is caused by crossing a bitangent ray in the other direction; see Figure 7(d).

The sensor will be called a *gap sensor*, as defined in [36], [62]. The name has the following motivation. Imagine sweeping radially to measure the distance to ∂E from (x_p, y_p). Every discontinuity in distance, as a function of angle, corresponds to a unique shadow component. Therefore, maintaining topological changes in $S(x_p, y_p)$ requires sensing the discontinuities, called *gaps*. The precise distance and angle is not needed; it is only assumed that as the pursuer moves it can track the gaps (in other words, as the gaps move over time, it knows the correspondence between previous gaps and current ones). For the split and merge events, it is furthermore assumed that the sensor indicates precisely which gaps were involved in the split or merge (for example, gaps a and b merged into c).

The gap sensor can then be used to define a filter that incrementally maintains the correct labels on the shadow components. If a component disappears, its label disappears along with it. If a component appears, it receives a 0 label because the area was just visible and the evader cannot be hiding there. If a component splits, the new components receive the same label. The final case is more interesting. If two components merge, then the new component receives a 1 label if *either* (or both) of the two components have a 1 label. Note that if the same components are involved in a merge followed by a split, then the labels may change from 0 and 1 to 1 and 1. Thus, the evader can find new hiding places after every merge.

We are now ready to describe a complete pursuit-evasion strategy based on the gap sensor:

Strategy 2 *Pursuit with the gap sensor*

Description: Assume that the pursuer has learned the cut ordering using the Strategy 1. A derived I-space $\mathcal{I}_{gap}$ and information transition function will now be described (recall (4)). At each stage, the following are recorded, as a derived I-state: 1) the position of the pursuer in the cut ordering, and 2) a label of 0 or 1 for each component of $S(x_p, y_p)$. As described above, the labels indicate whether each shadow component may contain the evader.

Initially, all shadow components (or gaps) receive 1 labels. The initial position, together with the label assignment, correspond to an element of $\mathcal{I}_{gap}$. The planning strategy proceeds by exhaustively exploring $\mathcal{I}_{gap}$. Consider traveling from any $\kappa(\eta_1) \in \mathcal{I}_{gap}$ to another $\kappa(\eta_2) \in \mathcal{I}_{gap}$. Based on the position in the cut ordering and the action that was applied, the next position in the cut ordering is known. Furthermore, based on the labels assigned in η_1, the pursuer can use the gap sensor to determine the resulting labels after moving to the new position. The strategy searches $\mathcal{I}_{gap}$ until it finds any I-state for which all labels are 0. The corresponding action sequence guarantees that the evader will be detected regardless of its initial position or motion. The complexity of the search method is exponential in the polygon size, in the same manner as for the algorithm in [23]; however, in practice, the implemented algorithm appears to terminate much more quickly, even on complicated examples (a behavior also observed in [23]).

Proposition 8 *The systematic search over $\mathcal{I}_{gap}$ of Strategy 2 finds a pursuit plan for the robot whenever one exists; otherwise, it reports failure after a finite number of steps.*

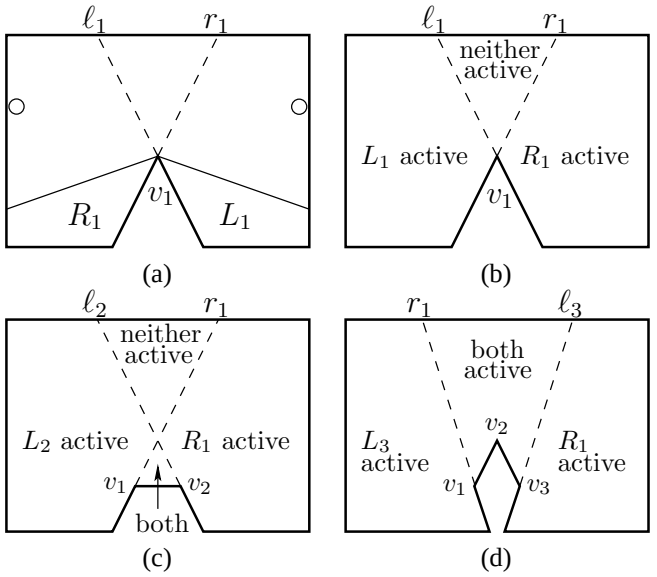
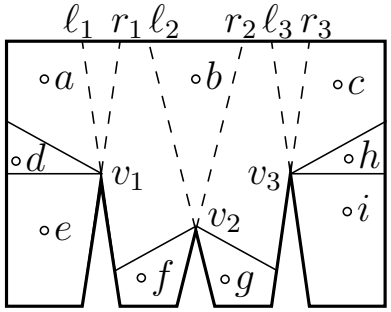


Fig. 8. (a) There are two primitives associated with every reflex vertex. In this example, R_1 is the primitive obtained after the robot crosses r_1 to the right. Likewise, L_1 is the primitive obtained after crossing ℓ_1 to the left. (b) Every reflex vertex i divides E into three regions, based on whether L_i, R_i, or neither is active. (c) and (d) show cases in which various left and right primitives are active



Position	Partition of A
a	$\{\{L_3\},\{L_2\},\{L_1\}\}$
b	$\{\{L_3\},\{R_1\}\}$
c	$\{\{R_3\},\{R_2\},\{R_1\}\}$
d	$\{\{L_3\},\{L_2,L_1\}\}$
e	$\{\{L_3,L_2,L_1\}\}$
f	$\{\{L_3,L_2\},\{R_1\}\}$
g	$\{\{L_3\},\{R_2,R_1\}\}$
h	$\{\{R_3,R_2\},\{R_1\}\}$
i	$\{\{R_3,R_2,R_1\}\}$

Fig. 9. The partitions of $A(x_p, y_p)$ are shown from nine different locations.

Proof: The set of possible positions in the cut ordering is finite. Furthermore, the set of all possible labelings is finite. Therefore, $\mathcal{I}_{gap}$ is finite. Systematic search explores every I-state in $\mathcal{I}_{gap}$ that is reachable from the initial state. Therefore, the strategy either finds a solution in $\mathcal{I}_{gap}$ or terminates in finite time after exhausting the reachable subset of $\mathcal{I}_{gap}$. If the method does not find a solution, then no solution exists because all possible action sequences are tried and the pursuit status is correctly maintained at every step. ■

C. Solution Using a Bitangent Sensor

Section VI-B described a clean solution to the pursuit-evasion problem; however, it is not fully satisfying because the gap sensor seems much more powerful than the sensors of Section III. This section considerably weakens the sensing assumption and nevertheless results in a complete strategy. The idea is to introduce a sensor that indicates split and merge information when a bitangent ray is crossed. This model is much closer to information that is inferred using the basic model from Section III. As shown in Section V, the robot can determine which inflection rays were crossed, but it cannot determine which bitangent rays were crossed without additional sensing. Section VI-E presents a pursuit-evasion strategy that works without sensing bitangent rays, but it remains open to show whether the strategy is complete.

Consider the set of all possible shadow components obtained by varying (x_p, y_p) over all of E. There is a finite total number of distinct shadow components. Figure 8 shows several cases that lead to what will be called *primitive* shadow components, or *primitives* for short. Every primitive corresponds to an inflection, as defined in Section VI-B. For each reflex vertex v_i, if it has a right inflection, the associated primitive is denoted by R_i. Likewise, if it has a left inflection, the primitive is L_i.

If $(x_p, y_p) \in E$ lies to the right of a right inflection, then the corresponding primitive is called *active*. Likewise, if (x_p, y_p) lies to the left of a left inflection, the corresponding primitive is also called active. Figure 8 shows cases in which various left and right primitives are active. Note that the complete set of primitives can be inferred from the cut ordering. Furthermore, the set $A(x_p, y_p)$ of primitives that are active from (x_p, y_p) can be determined from any position along the boundary of the cut ordering from the reading given by the touch sensor. Thus, $A(x_p, y_p)$ is known after the completion of any action RFOLLOW, LFOLLOW, or JUMP.

So far the discussion has characterized the appearance and disappearance of gaps from Section VI-B in terms of inflection crossings. These crossings can fortunately be inferred from the cut ordering. The next challenge is to characterize the effect of splitting or merging gaps now that we do not have a gap sensor. The result of splits and merges will be encoded as a partition of $A(x_p, y_p)$, which is denoted as $\pi(A)$. This is illustrated in Figure 9. Recall from Section VI-B that crossing a bitangent may cause shadow components to merge. A pair of primitives may merge into one component, which may eventually merge into another component. Any shadow component obtained by one or more merges is called a *compound*. Every compound can be uniquely described by listing all primitives that were merged to obtain it. See Figure 9.

The example in Figure 9 involves only *isolated* reflex vertices. For consecutive reflex vertices, the situation is slightly more complicated, but not problematic. Figure 10 shows an example in which there are two consecutive reflex vertices, v_1 and v_2. When the pursuer is in position a, primitive L_2

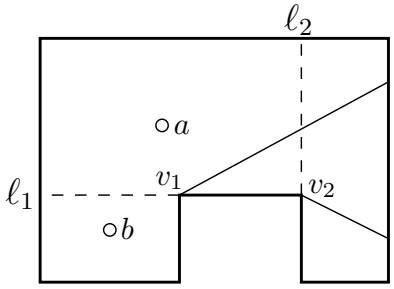


Fig. 10. An illustration of a sliding primitive.

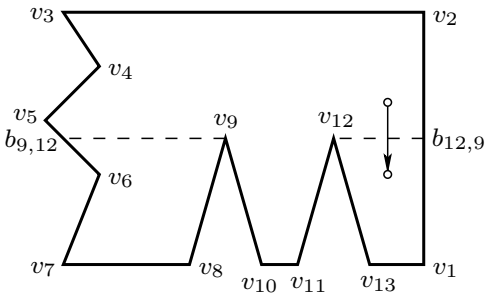


Fig. 11. When the pursuer crosses the bitangent ray, as shown in the right, the bitangent sensor indicates: 1) v_9 and v_{12} form the bitangent, 2) v_{12} is closer, and 3) the other bitangent ray ends between v_5 and v_6.

is active. However, when it passes to position b, note that the gap (boundary of the shadow component) makes a jump from vertex v_2 to v_1. This will be called a *sliding* primitive. The reflex vertex v_2, which generated L_2, is now in the interior of the shadow component. The shadow component that exists when the pursuer is at b is essentially the same component as L_2. Therefore, it can continue to be called L_2 (rather than changing its name to L_1). When using the gap sensor, this jump from v_2 to v_1 was in fact not even detectable.

Now consider keeping track of the pursuit status, as done in Strategy 2. As before, there is a label of 0 or 1 for each shadow component. In Section VI-B, the shadow component is expressed as a gap. Here, the shadow component is expressed as a set of primitives. Each shadow component is therefore expressed as a subset of $A(x_p, y_p)$, and all components of $S(x_p, y_p)$ together yield a partition of $A(x_p, y_p)$.

In this section, the gap sensor is replaced by a *bitangent sensor*. Unlike the gap sensor, it cannot detect the crossing of an inflection; this is closer to the models defined in Section III. Thus, the appearance or disappearance of a gap is not sensed, which is equivalent to being unable to sense whether a particular primitive becomes active or is deactivated. However, it is assumed that "perfect" information regarding a bitangent is sensed. In particular, whenever a bitangent ray is crossed, it is assumed that the pursuer immediately knows: the pair of reflex vertices, v_i, v_j that contribute to the bitangent, 2) which reflex vertex is closest, and 3) the location of the other bitangent ray endpoint in the cut ordering. See Figure 11.

This information is sufficient for determining which active primitives split or merge:

Proposition 9 *When a bitangent ray is crossed, the information provided by the bitangent sensor is sufficient for determining precisely which primitives split or merge.*

Proof: To determine which primitives split and merge, the following procedure can be followed. First, determine which of the two events (split or merge) of the primitives associated with v_i and v_j occurs. This can be done using the cut ordering by determining the direction in which a cut is crossed (clockwise or counterclockwise). Without loss of generality, assume that the bitangent ray crossed has endpoints at v_i and b_{ij}. Next, consider the two different intervals of ∂E with endpoints at v_j and b_{ji}. Choose the interval that does not contain v_i. Determine the active primitives associated with all of the reflex vertices lying in the interval. This is the first set of the active primitives participating in the current split or merge. The second set contains all of the active primitives lying in the intersection of an interval of ∂E between v_i and v_j and an interval between v_i and b_{ji}. Here choose the intervals that do not contain b_{ij}. The exact partition of the primitives within these sets into compounds is not determined yet, however, the two compounds resulting from the current split or merge is now determined. ■

A complete pursuit-evasion strategy can now be described:

Strategy 3 *Pursuit with the bitangent sensor*

Description: As in strategy 2, assume that the pursuer has learned the cut ordering. Consider the initial state. The set A_0 of initial active primitives is determined using the pursuer position in the cut ordering. The partition of $A(x_p, y_p)$ into compounds and primitives is not known initially, but this will not cause trouble. It can be assumed without harm that no primitives in $A(x_p, y_p)$ are merged into compounds. Every primitive is initially given a label of 1 to indicate that the corresponding shadow region might contain the evader.

Let A be the current active set, let $\pi(A)$ be the current partition, and let $l(\pi(A))$ be the assigned labels.

Suppose that the pursuer executes an action, LFOLLOW, RFOLLOW, or JUMP. At the end of the action, it uses the new position in the cut ordering to compute the new active set, A'. Any primitives that became active during the action execution are assigned 0 labels before considering any new merges. The detected bitangents are used to determine required splits and merges that are made when going from $\pi(A)$ to $\pi(A')$. Regarding the labels, the rules from Strategy 2 apply. The 0 label is preserved in a merge only if both components have the 0 label. The pursuer position in the cut ordering, together with A, $\pi(A)$, and $l(\pi(A))$, constitute a derived I-state. The update just described is the information transition function on a derived I-space, $\mathcal{I}_{bit}$.

Now that the information transitions have been determined, any systematic search can be used on $\mathcal{I}_{bit}$ to find an I-state in which all labels are 0.

Proposition 10 *The systematic search over $\mathcal{I}_{gap}$ of Strategy 3*

finds an strategy for the pursuer whenever one exists; otherwise, it reports failure after a finite number of steps.

Proof: If the strategy records the pursuit status in exactly the same way as Strategy 2, then clearly it is complete because both would systematically explore $\mathcal{I}_{gap}$. Although the new strategy does not directly use the gaps, labels are instead placed on elements of $\pi(A)$. Rather than maintaining the gap events, operations are maintained in primitives. The cut ordering indicates which inflections are crossed, and hence which primitives become active or non-active during execution; this is equivalent to indicating whether gaps appear or disappear. Using Proposition 9, the bitangent detector indicates which primitives split or merge, which is equivalent to knowing which gaps split or merge.

Using these equivalences, one complication remains: the initial compounds are not given. This can be handled by (incorrectly) assuming that there are no compounds. This implies that every active primitive can be assigned a unique label. Clearly this is not accurate if they truly belong to a compound that cannot be detected by the sensors initially. However, this is not a problem because all active primitives are initially assigned a 1 label. As they merge to form compounds, the resulting pursuit status is the same, with or without correctly obtaining the initial compounds. ■

D. Solution Using Pebble Visibility Sensor

An even less powerful, but nonetheless sufficient sensor is a *pebble visibility* sensor, $h_p(x) : X \rightarrow \{0, 1\}$. It is analogous to the evader detection sensor, yielding $h_p(x) = 1$ if and only if the robot position and the pebble position are mutually visible in the environment E. Such a sensor provides enough information to find all bitangents and determine the location of bitangent endpoints in the cut ordering, therefore allowing Strategy 3 to be executed without the bitangent sensor.

The approach proceeds by carefully studying the relative positions of points along ∂E. For any $s, t \in \partial E$, let (s, t) denote the open interval of ∂E obtained by traveling counterclockwise from s to t. Similarly, let $[s, t]$ denote the corresponding closed interval.

Let $F(E)$ denote the set of all reflex vertices of E. For any pair $v_i, v_j \in F(E)$, let $B(i, j)$ indicate whether there is a bitangent between v_i and v_j. Thus, B can be considered as a binary-valued function or logical predicate.

The following proposition establishes a necessary (but not sufficient) condition for $B(i, j)$:

Proposition 11 *For any $E \in \mathcal{E}$ and any $v_i, v_j \in F(E)$, if $B(i, j)$, then $v_i \notin (r_j, \ell_j)$ and $v_j \notin (r_i, \ell_i)$.*

Proof: If $B(i, j)$, then ∂E must be tangent to the line through v_i and v_j, precisely at v_i and v_j. If $v_i \in (r_j, \ell_j)$, then the line through v_i and v_j is not tangent at v_j (informally, when looking from v_i, there is no gap anchored at v_j). Similarly, if $v_j \in (r_i, \ell_i)$, then the line through v_i and v_j is not tangent at v_i. ■

For any pair, $v_i, v_j \in F(E)$, let $C(i, j)$ be a predicate indicating that they satisfy Proposition 11. If $C(i, j)$, then v_i and v_j are called a *bitangent candidate*. Note that $B(i, j)$ implies $C(i, j)$, but $C(i, j)$ does not necessarily imply $B(i, j)$. Why? Even though v_i and v_j are in the right positions along ∂E for a bitangent, they might not be mutually visible.

It will be convenient to make a notational convention regarding each pair $v_i, v_j \in F(E)$. Suppose $C(i, j)$ for some $v_i, v_j \in F(E)$. If $v_j \in [v_i, r_i]$, then the bitangent is called *right-handed*. If $v_j \in [\ell_i, v_i]$, then it is called *left-handed*. Note that if $v_j \in [r_i, \ell_i]$, then it cannot be a bitangent. If the bitangent is right-handed, then we can swap v_i and v_j to obtain one that is left handed; hence, we can always write it in a canonical way. From now on, assume that the pair v_i, v_j is always chosen so that the bitangent candidate is right-handed.

Consider the following strategy:

Strategy 4 *Locating bitangent endpoints*

Description: Suppose v_i and v_j form a right-handed bitangent candidate. To determine whether there is a bitangent, the robot navigates to v_j, DROPs the pebble, and navigates to v_i. If $C(i, j)$ and $h_p(x) = 1$, then $B(i, j)$.

To learn the position of $b_{i,j}$ in the cut ordering, the robot executes several LFOLLOW actions repeatedly. Immediately after it leaves v_i, $h_p(x)$ becomes 0. The endpoint $b_{i,j}$ is the first point after v_i, where $h_p(x) = 1$.

This allows the robot to find the position of $b_{i,j}$ relative to the vertices of E, but not to the cut endpoints. To determine the exact ordering, the robot moves to each cut endpoint c that lies on the same edge as $b_{i,j}$ and executes the RFOLLOW action. If at any moment during this step (including the departure point, c) $h_p(x) = 1$, then $b_{i,j}$ is on the right of c; otherwise, $b_{i,j}$ is on the left of c.

By replacing LFOLLOW with RFOLLOW and vice versa, the same method can be used to determine the position of $b_{j,i}$. Repeating these steps for all candidate bitangents, the robot can learn the positions of all bitangent endpoints.

Proposition 12 *Using Strategy 4 the pursuer learns the positions of bitangent endpoints in the cut ordering.*

Proof: The method used in Strategy 4 to find the bitangent endpoints is based on two facts.

The first is that for any $s \in (b_{i,j}, v_i)$, points s and v_j are not mutually visible. Otherwise, the segment from s to v_j would intersect the bitangent line in two places, between $b_{i,j}$ and v_i, and also at v_j, which is impossible.

The second fact is that there exists $t \in (v_j, b_{i,j})$ such that for all $s \in (t, b_{i,j})$, the points s and v_j are mutually visible. The existence of such t follows from the general position assumption.

■

The information provided by Strategy 4 is enough to generate the same output as the bitangent sensor did in

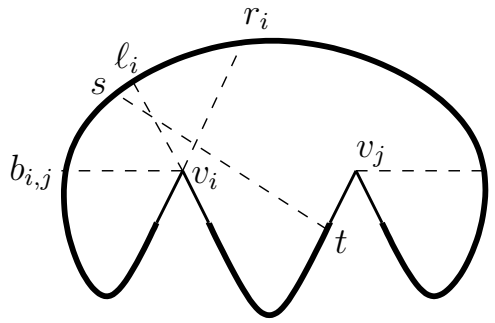


Fig. 12. For any mutually visible points $s, t \in \partial E$, for which $t \in (v_i, v_j)$ and $s \in (\ell_i, v_i)$, it follows that $s \in (\ell_i, b_{i,j})$, and that $b_{ij} \in (s, v_i)$.

Section VI-C. Therefore, the following strategy has the same property of completeness as Strategy 3:

Strategy 5 *Pursuit with the pebble visibility sensor*

Execute Strategy 4 to locate bitangent endpoints. Next, execute Strategy 3 using obtained information instead of a bitangent sensor to determine split and merge events.

E. Solution With No Special Sensors

Now we return to the original sensing model, which was presented in Section III. The only additional sensor is the simple binary detection sensor of Section VI-A. It would be convenient to follow the approach of Strategy 3; however, without the bitangent sensor or pebble sensor, the pursuer is unable to obtain information about split and merge events. Nevertheless, based on information in the cut ordering, the pursuer can reason about where bitangents *might* be. For such candidates, the pursuer also constructs an *approximation* to the bitangent ray endpoints. Using this approach, the pursuer pretends that it receives all necessary bitangent information and applies a strategy similar to Strategy 3.

Suppose $C(i,j)$ for some $v_i, v_j \in F(E)$. If $B(i,j)$, then where could the bitangent endpoints $b_{i,j}$ and $b_{j,i}$ possibly lie along ∂E? It will be important to make a conservative approximation. Upper bounds will be determined on their locations. A simple conservative bound is given by the following:

Proposition 13 *For any $E \in \mathcal{E}$ and any $v_i, v_j \in F(E)$, if $B(i,j)$, then $b_{i,j} \in [\ell_i, v_i] \cap [\ell_j, v_i]$ and $b_{j,i} \in [v_j, r_i] \cap [v_j, r_j]$.*

Proof: If $b_{i,j}$ appears before ℓ_i, then an edge incident to v_i must be at least partially visible from v_j, which contradicts the assumption $B(i,j)$. Similarly, if $b_{i,j}$ appears before ℓ_j, then an edge incident to v_j must be at least partially visible from v_i. Similar arguments apply for $b_{j,i}$. ■

Using information from the cut ordering, a tighter bound on the location of $b_{i,j}$ can be obtained. See Figure 12.

Proposition 14 *For any $E \in \mathcal{E}$, any $v_i, v_j \in F(E)$, and any mutually visible pair of points $s, t \in \partial E$ such that $s \in [\ell_i, v_i]$ and $t \in [v_i, v_j]$, if $B(i,j)$, then $b_{i,j} \in [s, v_i]$.*

Proof: The proposition follows from the simple fact that the segment from s to t must intersect the bitangent line somewhere between v_i and v_j. This implies that s must hit ∂E before $b_{i,j}$. Otherwise, the segment from s to t would intersect the bitangent line in two places, which is geometrically impossible. ■

Similarly, there is a symmetric equivalent that corresponds to the other bitangent endpoint, $b_{j,i}$:

Proposition 15 *For any $E \in \mathcal{E}$, any $v_i, v_j \in F(E)$, and any mutually visible pair of points $s, t \in \partial E$ such that $s \in [v_j, r_j]$ and $t \in [v_i, v_j]$, if $B(i,j)$, then $b_{j,i} \in [v_j, s]$.*

Now we can use Propositions 13 to 15 to obtain approximate locations of $b_{i,j}$ and $b_{j,i}$. Let these be denoted as $\hat{b}_{i,j} \in \partial E$ and $\hat{b}_{j,i} \in \partial E$, respectively. Applying Proposition 13, we obtain an initial approximation of $\hat{b}_{i,j} = \ell_i$ or $\hat{b}_{i,j} = \ell_j$, depending on which is closest to v_i in counterclockwise order. Similarly, we obtain $\hat{b}_{j,i} = r_i$ or $\hat{b}_{j,i} = r_j$, whichever is reached first after traveling counterclockwise from v_j.

These approximations can be improved by looking for any cuts that satisfy Proposition 14. Each cut is a candidate for the pair s, t, if either the reflex vertex or the cut endpoint lies in (v_i, v_j). Among all cuts that satisfy Proposition 14, pick the cut for which s is closest for v_i in counterclockwise order. In this case, let $\hat{b}_{i,j} = s$. Similarly, $\hat{b}_{j,i}$ can be obtained by applying Proposition 15 on every possible cut. With these approximations, we now state the strategy for our original sensor model:

Strategy 6 *Pursuit with the pebble and the contact sensors*

Description: Strategy 3 is executed by assuming that $C(i,j)$ implies $B(i,j)$ every time in the worst case and by using $\hat{b}_{i,j}$ and $\hat{b}_{j,i}$ instead of $b_{i,j}$ and $b_{j,i}$.

To argue the correctness of Strategy 6, first we introduce the following lemma:

Lemma 16 *For a single action* LFOLLOW, RFOLLOW, *or* JUMP, *the labeling of shadow components in Strategy 3 is invariant with respect to the order in which inflections and bitangents are crossed.*

Proof: First we note that any appear or disappear event not involved in a split or merge can be placed in any order without affecting the labeling. Likewise, any disjoint merges or splits can be swapped.

Since the RFOLLOW, LFOLLOW, and JUMP motions each produce a linear motion, it is impossible to cross the same bitangent ray or inflection more than once. Therefore, appear and disappear events of the same primitive, as well as split

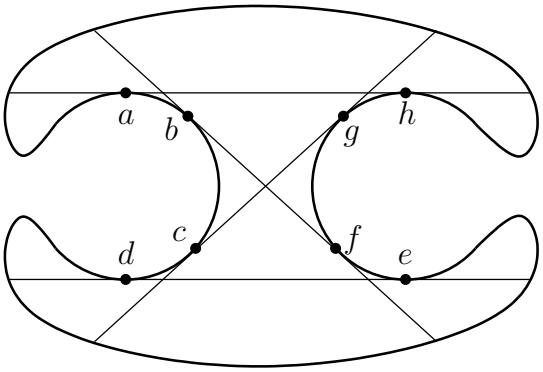


Fig. 13. Between any pair of reflex chains, there are at most four bitangents, and the contributing reflex vertices are sorted from a to h along ∂E as shown.

and merge events of the same compound can not occur simultaneously after execution of a single action. Consider now a situation in which a single merge or split appears together with the appear or disappear event of the same primitive. Due to the geometry of the inflections and bitangents used in Proposition 14, the appear or disappear event must occur before the merge event, and the appear or disappear event must occur after the split event. This guarantees that the order of such events is fixed, and can not affect the labeling.

Now consider there are multiple splits and/or merges which occur during a single action. The order of multiple splits does not matter; the same label propagates to the final components. Similarly, the order of multiple merges does not matter because all resulting components will share the same label in the end. The only difficulty appears if multiple merges occur together with multiple splits of the compounds consisting of the same primitives. However, this is not possible, since an approximation of a bitangent ray can cross a straight line only once. Thus regardless of the crossing order, the resulting partition $\pi(A)$, after applying the action, is invariant. ■

Proposition 17 *If Strategy 6 finds a pursuit plan, then the strategy is correct.*

Proof: In the current setting, the order in which inflections and bitangents are crossed while executing a single action, such as LFOLLOW, cannot be determined. Lemma 16 is useful here, since it states that the labeling is invariant with respect to this order. Furthermore, all events due to inflection crossings are detected, as in Strategy 3. The only danger of having an incorrect plan is therefore associated with bitangents. With a perfect bitangent detector, we have Strategy 3, which always returns correct plans. In the current setting, merges may potentially be applied too liberally. For each $C(i,j)$ a merge is performed that approximates conservatively the set of potential locations for the evader. This implies that if a plan forces all labels to zero, then the evader cannot escape detection. ■

The only remaining question is whether the strategy is complete: Does it return a solution for any cut ordering for which there exists a guaranteed solution? The trouble with establishing completeness is that we have to consider all possible environments that could be realized from a cut ordering. Since we have made a conservative approximation to bitangents, we must consider worst-case environment that realize as many bitangents as possible. Is it possible that for any cut ordering, all bitangent candidates are realized? This was an open conjecture in [65], and the following proposition implies that the conjecture is false:

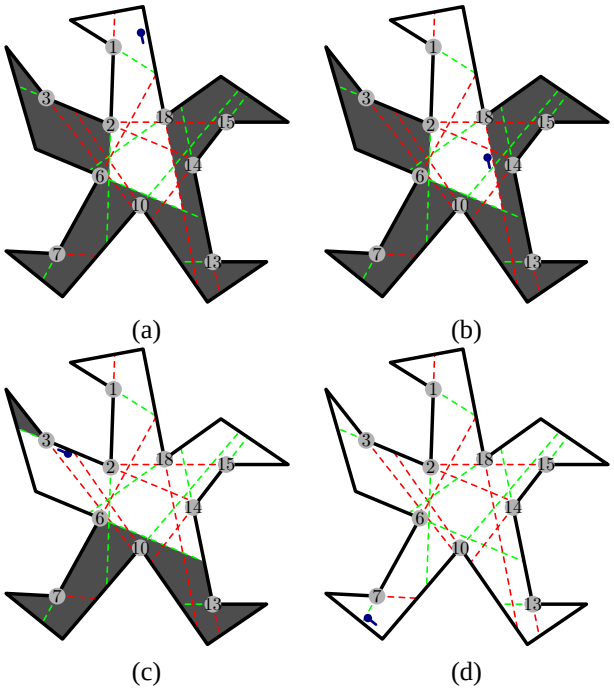


Fig. 14. Computed example for pursuit-evasion. Shaded regions are shadow regions in which an evader might be. Note that from (a) to (b), the robot travels along the boundary of a shadow region, nevertheless the labeling of the shadow region cannot be updated until the action ends. Figures (c) and (d) show the last steps of the plan.

Proposition 18 *Between any pair of reflex vertex chains, there are at most four bitangents.*

Proof: See Figure 13. Consider two mutually visible discs, which are approximated by numerous tiny edges and reflex vertices. Along the reflex chain of one disc, at most one of the left cuts can be tangent to the other disc. Likewise, at most one right cut can be tangent to the other disc. By symmetry, there are at most two more bitangents by considering left and right cuts from the second disc to the first one. ■

Thus, there may be numerous bitangent candidates that do not produce actual bitangents. It remains an interesting open question to establish completeness of the strategy. We have been unable to construct an example for which a pursuit plan exists and the algorithm is unable to solve it.

The pursuit-evasion strategy was implemented in simulation. Two computed examples are shown in Figures 14, and 15. Using only the cut ordering and the strategy for pursuit-evasion, the robot generates the plan for finding all of the

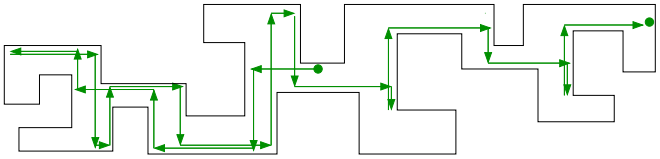


Fig. 15. Computed solution path for detecting the evaders.

evaders in the environment. The computed solution paths are shown[3].

VII. Conclusions

This paper has developed and analyzed I-spaces associated with a simple robot that follows walls, jumps from reflex vertices, and carries a pebble. Each of Sections IV to VI presented problems that were progressively more complicated. In Section IV, simple I-spaces arose from a robot that can only follow walls and sense a fixed pebble. In that case, the robot can count the number of vertices and how many times it wrapped around the polygon; however, without the pebble we proved that it cannot even accomplish these tasks. In Section V, the robot gained the ability to classify the vertex type, jump from reflex vertices, and also move a pebble. The cut ordering was introduced as the precise characterization of what can be learned about the environment under this model. A strategy for learning the cut ordering was presented and the method was proved to be complete in the sense that no further information about the environment can possibly be acquired by the robot. In Section VI, the robot was equipped with an additional sensor that enabled it to detect any evaders that are within its field of view. Assuming the existence of sensors that can determine bitangent structure, complete pursuit-evasion strategies were presented in Sections VI-B and VI-C. Even without such sensors, Section VI-D introduced a complete pursuit-evasion algorithm that only needs to detect pebbles using the same visibility sensors that detects evaders, rather than requiring direct bitangent sensing. Finally, when we do not even give robot this ability, Section VI-E presented a pursuit-evasion strategy that computes plans that are guaranteed to find any evaders; however, it remains an open problem to prove its completeness.

Many other open questions and possible future research directions remain. In terms of information spaces, two general directions are: 1) developing filters, and 2) planning in I-spaces. It is important to develop minimalist, *combinatorial* filters that incrementally maintain small amounts of necessary information using I-maps and derived I-spaces. These compute important statistics to solving tasks, but do not tell the robot how to move. Furthermore, these do not need to perform state estimation, as in classical filtering. Once such filters are developed, the challenge is to develop planning strategies that manipulate derived I-states to accomplish some task. The remainder of this section presents interesting filtering and planning questions that extend naturally from the work presented in this paper.

Consider the tasks that were solved across Sections IV to VI. In what minimal ways does the robot model need to be extended to accomplish each of these tasks for the case in which E is a polygonal region with arbitrarily many holes? What if the number of holes is fixed in advance? Some recent work that uses models similar to ours shows how to count holes in such environments [59].

Numerous models can be studied by allowing uncertainties in actuation and/or sensing. For example, what if there are known probabilities associated with misclassifying vertices? When the robot jumps, what if it does not precisely move in a direction parallel to the edge from which it departs? What if the robot cannot even move in a straight line?

Another direction involves departing from polygonal models. In some sense, there is nothing particularly special about vertices. What kinds of models and solution can be developed in the case of a smooth environment? What if the environment is piecewise smooth? In such settings, we could use additional markers or landmarks that could be arbitrarily placed in the environment. Where do landmarks need to be placed and what needs to be sensed about them to accomplish to learn the structure of the environment or perform pursuit-evasion tasks?

Some standard questions arise, which are straightforward to formulate, but extremely challenging to address. What happens when the robot is modeled as a rigid body, as opposed to a point? This brings configuration space obstacles into the analysis. In the simplest case, the robot may be a disc, which yields symmetries with respect to robot orientation. More generally, the robot may be a rotating polygonal body. A three dimensional version of the problems presented in this paper can also be posed. In this case, we would be confronted with the known complexity of three-dimensional visibility computations [48], [54]. To further complicate matters, wall following obtains a second degree of freedom; how can the robot be forced to reach a particular vertex?

Numerous complexity questions arise in the context of this work. As sensing and actuation become simpler, how does the complexity increase in terms of the number of actions and the amount of computation? What are the precise upper and lower complexity bounds for accomplishing the tasks in this paper? Understanding tradeoffs between sensing, actuation, and computation are crucial to the development of robotic systems that use reduced amounts of sensing and actuation.

Finally, there is the important connection between the presented work and the development of robotic systems that can accomplish tasks with less information. The models used here are inspired by the success of commercial systems such as the Roomba vacuum cleaning robot. However, substantial work remains to adapt the models and strategies presented in this paper. What adaptations to the models are most appropriate in experimental robot systems? What kinds of failures must be accounted for in practice? Can versions that take probabilistic sensing errors into account be developed?

[3]This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. This includes five AVI format movie clips, which show robot executing pursuit plans for various environments

REFERENCES

[1] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory, 2nd Ed.* Academic, London, 1995.

[2] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proc. Annual Symposium on Foundations of Computer Science*, 1998.

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. I, 2nd Ed.* Athena Scientific, Belmont, MA, 2001.

[4] D. Bienstock and P. Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12:239–245, 1991.

[5] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrains. In *Proc. ACM Symposium on Computational Geometry*, pages 494–504, 1991.

[6] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. Annual Symposium on Foundations of Computer Science*, pages 132–142, 1978.

[7] J. Castellanos, J. Montiel, J. Neira, and J. Tardós. The SPmap: A probabilistic framework for simultaneous localization and mapping. *IEEE Transactions on Robotics & Automation*, 15(5):948–953, 1999.

[8] W.-P. Chin and S. Ntafos. Optimum watchman routes. *Information Processing Letters*, 28:39–44, 1988.

[9] D. Crass, I. Suzuki, and M. Yamashita. Searching for a mobile intruder in a corridor – The open edge variant of the polygon search problem. *International Journal Computational Geometry & Applications*, 5(4):397–412, 1995.

[10] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment I: The rectilinear case. Available from http://www.cs.berkeley.edu/∼christos/, 1997.

[11] G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics & Automation*, 17(3):229–241, 2001.

[12] B. R. Donald. The complexity of planar compliant motion planning under uncertainty. In *Proc. ACM Symposium on Computational Geometry*, pages 309–318, 1988.

[13] B. R. Donald. On information invariants in robotics. *Artificial Intelligence Journal*, 72:217–304, 1995.

[14] B. R. Donald and J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. In *Proc. IEEE International Conference on Robotics & Automation*, pages 190–197, 1991.

[15] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[16] M. A. Erdmann. On motion planning with uncertainty. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, August 1984.

[17] M. A. Erdmann. Randomization for robot tasks: Using dynamic programming in the space of knowledge states. *Algorithmica*, 10:248–291, 1993.

[18] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Transactions on Robotics & Automation*, 4(4):369–379, August 1988.

[19] S. P. Fekete, R. Klein, and A. Nüchter. Online searching with an autonomous robot. In *Proc. Workshop on Algorithmic Foundations of Robotics*, Zeist, The Netherlands, July 2004.

[20] Y. Gabriely and E. Rimon. Competitive complexity of mobile robot on line motion planning problems. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 249–264, 2004.

[21] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[22] K. Y. Goldberg and M. T. Mason. Bayesian grasping. In *Proc. IEEE International Conference on Robotics & Automation*, 1990.

[23] 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.

[24] L. Guilamo, B. Tovar, and S. M. LaValle. Pursuit-evasion in an unknown environment using gap navigation trees. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

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

[26] T. Kameda, M. Yamashita, and I. Suzuki. On-line polygon search by a seven-state boundary 1-searcher. *IEEE Transactions on Robotics*, 2006. To appear.

[27] I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. *IEEE Transactions on Robotics & Automation*, 13(6):814–822, December 1997.

[28] I. Kamon, E. Rivlin, and E. Rimon. Range-sensor based navigation in three dimensions. In *Proc. IEEE International Conference on Robotics & Automation*, 1999.

[29] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 441–447, 1993.

[30] J. M. Kleinberg. On-line algorithms for robot navigation and server problems. Technical Report MIT/LCS/TR-641, MIT, Cambridge, MA, May 1994.

[31] H. W. Kuhn. Extensive games and the problem of information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, pages 196–216. Princeton University Press, Princeton, NJ, 1953.

[32] P. R. Kumar and P. Varaiya. *Stochastic Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[33] Y. Landa and R. Tsai. Visibility of point clouds and exploratory path planning in unknown environments. *Communications in Mathematical Sciences*, 6(4):881–913, December 2008.

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

[35] S. L. Laubach and J. W. Burdick. An autonomous sensor-based path-planning for planetary microrovers. In *Proc. IEEE International Conference on Robotics & Automation*, 1999.

[36] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.

[37] S. M. LaValle, D. Lin, L. J. Guibas, J.-C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *Proc. IEEE International Conference on Robotics & Automation*, pages 737–742, 1997.

[38] 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.

[39] S. Lazebnik. Visibility-based pursuit evasion in three-dimensional environments. Technical Report CVR TR 2001-01, Beckman Institute, University of Illinois, 2001.

[40] J.-H. Lee, S. Y. Shin, and K.-Y. Chwa. Visibility-based pursuit-evasions in a polygonal room with a door. In *Proc. ACM Symposium on Computational Geometry*, 1999.

[41] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[42] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[43] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. In *Proc. ACM Symposium on Theory of Computing*, pages 322–333, 1988.

[44] B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted graphs. *Theoretical Computer Science*, 58:209–229, 1988.

[45] L. Murphy and P. Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *Proceedings IEEE International Conference on Robotics & Automation*, 2008.

[46] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *1st International Conference on Artificial Intelligence*, pages 509–520, 1969.

[47] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, 1987.

[48] J. O'Rourke. Visibility. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 643–663. Chapman and Hall/CRC Press, New York, 2004.

[49] G. Owen. *Game Theory*. Academic, New York, 1982.

[50] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[51] 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, Dept. of Computer Science, KAIST, Seoul, South Korea, January 2001.

[52] R. Parr and A. Eliazar. DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *Proc. International Joint Conference on Artificial Intelligence*, 2003.

[53] 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.

[54] M. Pocchiola and G. Vegter. The visibility complex. *International Journal Computational Geometry & Applications*, 6(3):279–308, 1996.

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

[56] N. Rao, S. Kareti, W. Shi, and S. Iyenagar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410:1–58, Oak Ridge National Laboratory, July 1993.

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

[58] B. Simov, S. M. LaValle, and G. Slutzki. A complete pursuit-evasion algorithm for two pursuers using beam detection. In *Proc. IEEE International Conference on Robotics & Automation*, pages 618–623, 2002.

[59] S. Suri, E. Vicari, and P. Widmayer. Simple robots with minimal sensing: From local visibility to global geometry. *International Journal of Robotics Research*, 27(9):1055–1067, September 2008.

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

[61] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[62] B. Tovar, R Murrieta, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518, June 2007.

[63] M. Yamashita, H. Umemoto, I. Suzuki, and T. Kameda. Searching for a mobile intruder in a polygonal region by a group of mobile searchers. *Algorithmica*, 31:208–236, 2001.

[64] B. Yamauchi, A. Schultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In *Proc. IEEE International Conference on Robotics & Automation*, pages 3715–3720, 2002.

[65] A. Yershova, B. Tovar, R. Ghrist, and S. M. LaValle. Bitbots: Simple robots solving complex tasks. In *AAAI National Conference on Artificial Intelligence*, 2005.

Max Katsev received B.S. and M.S. degrees in applied mathematics from the Kyiv National University, Kyiv, Ukraine, in 2007 and 2009 respectively. He is currently working toward the Ph.D. degree with the Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests include planning algorithms, control theory, artificial intelligence, and sensing.

Anna Yershova earned the B.S. degree in applied mathematics from Kharkiv National University in 1999, the M.S. degree in computer science from Iowa State University in 2003, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 2008. She is currently a postdoctoral researcher in the Department of Computer Science at Duke University. Her research interests include structural computational biology, algorithmic robotics, computational geometry, and motion planning.

Benjamin Tovar is currently a postdoctoral research fellow at the Department of Mechanical Engineering at Northwestern University. He holds the degrees of Ph.D. in Computer Science (2009), and M.S. in Electrical Engineering (2004), both from the University of Illinois at Urbana-Champaign. His main research interest is robotics, with a special focus on understanding the information requirements for solving robotic tasks to endow artificial systems with autonomy under severe sensing limitations, high actuation uncertainty, and limited computational power.

Robert Ghrist is the Andrea Mitchell University Professor of Mathematics and Electrical & Systems Engineering at the University of Pennsylvania. Ghrist is a (2004) PECASE-winning mathematician and a Scientific American SciAm50 winner (2007) for leadership in research. His specialization is topology in applied mathematics.

Steven M. LaValle received the B.S. degree in computer engineering, and the M.S. and Ph.D. degrees in electrical engineering, from the University of Illinois at Urbana-Champaign in 1990, 1993, and 1995, respectively. From 1995-1997 he was a postdoctoral researcher and lecturer in the Department of Computer Science at Stanford University. From 1997-2001 he was an Assistant Professor in the Department of Computer Science at Iowa State University. He is currently a Professor in the Department of Computer Science at the University of Illinois. His research interests include planning algorithms, sensing, motion planning, computational geometry, and control theory. He authored the book Planning Algorithms, Cambridge University Press, 2006.