Extracting Visibility Information by Following Walls

Anna Yershova, Benjamín Tovar, and Steven M. LaValle

Department of Computer Science, University of Illinois, Urbana, IL 61801 USA
{yershova,btovar,lavalle}@uiuc.edu

Summary. This paper presents an analysis of a simple robot model, called Bitbot. The Bitbot has limited capabilities; it can reliably follow walls and sense a contact with a wall. Although the Bitbot does not have a range sensor or a camera, it is able to acquire visibility information from the environment, which is then used to solve a pursuit-evasion task. Our developments are centered on the characterization of the information the Bitbot acquires. At any given moment, due to the sensing uncertainty, the robot does not know the current state. In general, uncertainty in the state is one of the central issues in robotics; therefore, the Bitbot model serves as an example of how the notion of information space naturally handles uncertainty. We show that state estimation with the Bitbot is a challenging problem, related to the well-known open problem of characterizing visibility graphs in computational geometry. However, state estimation becomes unnecessary to the achievement of the Bitbot's visibility tasks. We show how pursuit-evasion strategy is derived from a careful manipulation with histories of observations, and present analysis of the algorithm and experimental results.

1 Introduction

This paper analyzes the capabilities of a very simple robot, the Bitbot, which has severe sensors and actuators limitations. In fact, the Bitbot can only follow walls, and indicate if it is in contact with a wall or not. The capabilities of the robot are so limited that a precise knowledge of its state is unattainable. Nevertheless, this paper shows that the precise knowledge of the state is not needed for the Bitbot to complete some complicated robotic tasks. Particularly, the Bitbot can generate an environment representation from which visibility information can be inferred. This information is obtained without any range sensor, a camera, or any type of odometry.

The study for such simple robots is motivated by our interest in dealing with uncertainty in robotic systems. Classically, uncertainty is not addressed directly during the design of algorithms. During execution, it is trusted that the state estimator will produce an error small enough so that the estimated state can be taken as the current state. While this approach is adequate for several robotic tasks, it does not incorporate the inherent uncertainty in the design of the algorithm. Moreover, as it is the case for the Bitbot, an estimate of the state is by itself a very difficult open problem. It would seem that such lack of a state estimate makes the Bitbot quite useless.

Instead of focusing on the state estimation, we focus the algorithm development on particular tasks, so that a robot like the Bitbot can solve complicated tasks. The key idea is to incorporate the uncertainty in the design of the particular algorithms through an *information space*. In this case, the algorithms manipulate information states until a certain goal information state is reached. Instead of operating on current state estimates, the algorithm operates on the histories of observations and actions performed by the robot. For a full development of information spaces the reader is referred to [9]. Such approach has been used before in the context of manipulation [3, 4], for example, by orienting polygonal parts with a sensorless system [1, 5, 6]. Information spaces also appear (sometimes implicitly) in algorithms for robot localization [2]. A careful study of such information spaces allows very limited robots to localize themselves in polygonal environments [10, 11].

Since an information space encodes the state of the *whole* robotic task, rather than only the state of the robot, it provides natural description for the visibility-based pursuit-evasion problem. In this task the pursuer tries to visually detect an evader, which in turn tries actively to hide. The problem here is the uncertainty in the position of the evader, and the pursuer should move to reduce such uncertainty. A strategy was presented in [7] for a pursuer robot with perfect knowledge of a particular environment, and with omnidirectional, infinite range vision. When the environment is not known this imperfect information adds to the uncertainty of the state. Examples of such scenarios were studied in [8, 13]. Here, not only the pursuers did not know the environment, but had severe sensor limitations.

In the present paper we develop a full analysis of the information space of the Bitbot. We have shown in [14] that a single Bitbot placed in an unknown environment can learn enough information about the environment to localize itself with respect to a special representation, called the *cut diagram*. The cut diagram stores information about all of the reflex and convex vertices together with the extensions from reflex vertices to the edges in the polygon. We also briefly presented the strategy for locating evaders by a Bitbot. The current paper presents a first and complete analysis of the visibility information that is gathered with the limited sensing capabilities of a Bitbot. One of the goals of this work is to show what visibility information can be inferred from the cut diagram and how a Bitbot can use this information to solve visibility tasks.

The problem of finding environments consistent with a cut diagram is similar to that of finding polygons consistent with a given visibility graph [12]. This has been long known as an open problem in computational geometry. However, in this paper we continue to pursue an idea that to solve a task with a robot it is not necessary to estimate the state. That is, to find all of the evaders in the environment it is not necessary to know the exact geometry of a polygon the robot is in. Partial information, in this case the cut diagram of the polygon, is enough.

2 The Model

The Bitbot is modeled as a point moving in the polygonal environment $E \in \mathcal{E}$, in which E is a bounded, open set in $\mathbb{R}^2$, and the boundary ∂E is assumed a simple closed and connected polygonal curve. The set $\mathcal{E}$ is the set of all such environments. The circular sequence of vertices $V^E = \{v_i \mid i = 0, \ldots, n-1\}$ along ∂E is assumed to be clockwise ordered. A vertex of ∂E is called *reflex* if its incident edges form an angle strictly greater than π. We denote the set of reflex vertices of ∂E with V_r^E. A vertex in $V^E \setminus V_r^E$ is called *convex*.

The Bitbot can choose among two types of movements. First, it can follow the walls in either direction. Second, when approaching a reflex vertex, it can choose to go straight of the reflex vertex following a straight line, until ∂E is reached again. This straight line has the same slope as the last edge followed in the polygon before arriving at the reflex vertex. The Bitbot is equipped with a contact sensor, which indicates whether or not there is a contact with the boundary of the environment. These movements and sensor capabilities allow a Bitbot to differentiate between the situations in which it is in the contact with the walls or in which it is traveling in the interior of the environment. With this information it can be easily determined whether the Bitbot is in contact with a reflex vertex or not. During the present analysis, we assume that the contact sensor is powerful enough to detect convex vertices. The analysis still holds if this assumption is removed, with a natural increase in the uncertainty of the Bitbot state[1].

Given two vertices $u, v \in \partial E$, assume that u and v are consecutive in V_E. Furthermore, assume that u is a reflex vertex. Suppose, that the Bitbot has just visited v, traveled towards u, and at u it decided to keep straight with the same slope as the edge $\overline{uv}$. Call u_l the point the Bitbot hits in ∂E (Figure 1.a). The segment $\overline{uu_l}$ is called a *cut* at u, with u_l being the cut's endpoint. It is said that the cut is incident to the edge of ∂E containing u_l. If the environment E is divided into two parts along the cut at u, the vertex u does not belong to the polygon containing the vertex v. Instead, u lies on the edge $\overline{vu_l}$. We call this polygon a *cave* of the cut at u. Note, that there are exactly two cuts per reflex vertices. If u precedes v in the order given in V_E, the polygon containing v is called the *left cave* at u, and the cut $\overline{uu_l}$ is called the *left cut*. Otherwise, if u follows v, the *right cave* and the *right cut* $\overline{uu_r}$ at u are obtained. An example of a polygon with the set of all of its cuts is shown

[1] Incidentally, the name Bitbot originated in the absence of this assumption. Loosely speaking, the sensing is reduced to one bit of information that indicates whether the robot is in contact with a wall or not.

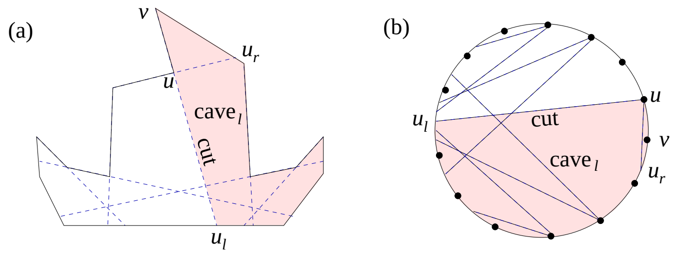

Fig. 1. (a) A polygon and all of its cuts are shown. For the left cut $\overline{uu_l}$ its left cave is shaded. (b) The corresponding cut diagram.

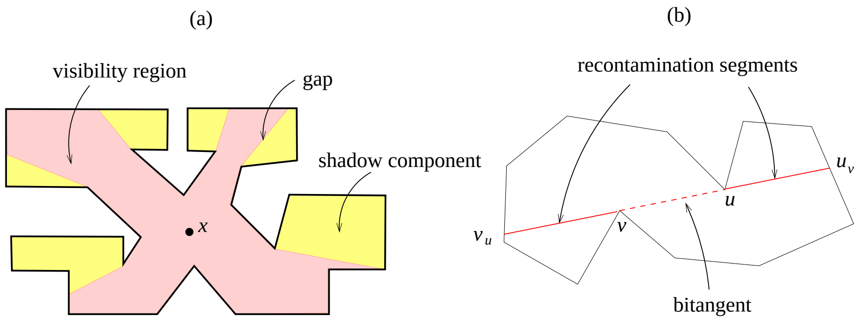

Fig. 2. (a) Visibility region, $V(x)$, shadow region, $e \setminus V(x)$, and gaps for the point x are shown. (b) The bitangent $\overline{uv}$ between the two reflex vertices u and v, if extended outward generates two bitangent complements, $\overline{uu_v}$ and $\overline{vv_u}$, correspondingly.

on Figure 1.a. We make the general position assumption that no three vertices of ∂E lie in the same line, thus, the point u_l cannot be a vertex itself. The set of all of the cuts is denoted as C^E, and the set of all of the cuts endpoints is denoted as V_c^E. The state space of the robot is $X = \mathbb{R}^2 \times S^1$. Given that the Bitbot can only recognize if it is in contact with a wall, at a vertex, or at an endpoint of a cut, the state space can be collapsed into a discrete set of states, namely $X_d = V^E \cup V_c^E$.

For any point $x \in E$, let $V(x) \subset E$ be the *visibility region* of x, that is, the set of all of the points $y \in E$ such that the line segment $\overline{xy}$ does not intersect ∂E (see Figure 2.a). Call $E \setminus V(x)$ the *shadow region*, which is the set of all of the points in E that are not visible from x. There may be several connected components in the shadow region. Each of them can be associated with some reflex vertex in the environment. We call each of them *a shadow component.*

Consider now two mutually visible reflex vertices, u and v, in the environment E. It is said that the segment $\overline{uv}$ is a *bitangent* if it can be extended outwards from both endpoints without lying outside the polygon. Now consider the two maximal extensions, $\overline{uu_v}$ and $\overline{vv_u}$, of the bitangent $\overline{uv}$. These segments are called *bitangent complements* at u and v respectively (see Figure 2.b).

3 Cut Diagram

In this section we consider the *cut diagram* data structure, the environment representation the Bitbot generates. The method for constructing the cut diagram by the Bitbot is described in [14]. In the following sections, we explain how the Bitbot uses it to recover visibility information. The cut diagram represents the adjacencies of vertex and edges of ∂E together with all the cut incidences of the polygon. Formally, the cut diagram is defined as follows:

Definition 1. *A tuple $D = (s, V, C)$ is called a cut diagram of a polygon E iff*

- *s is a simple closed curve.*
- *$V \subset s$ is a finite set of points.*
- *Each $c \in C$ is a line segment with endpoints in V, such that it intersects s at exactly two points (the endpoints).*
- *There is a bijection $g_v : V^E \cup V_c^E \rightarrow V$, such that the clockwise ordering of points along δE is preserved along the curve s.*
- *There is $\overline{\text{a bijection } g_c : C^E \rightarrow C, \text{ such that for a cut } c = \overline{uv} \in C, g_c(\overline{uv}) = g_v(u)g_v(v)}$.*

Thus, a polygon together with its cuts is itself a cut diagram. A cut diagram can be drawn canonically as follows. Let s to be the unit circle, and choose V as $|V^E \cup V_c^E|$ equidistant points on s. The cuts incidences are then added in the natural way as chords in the circle, choosing their endpoints in the same clockwise order in the incident edge. An example of a cut diagram corresponding to a polygon is shown on Figure 1.b. From now on we refer to a circle arc $\widehat{v_1 v_2}$ as the arc on the circle s connecting vertex v_1 and vertex v_2, which corresponds to edge $\overline{v_1 v_2}$ in ∂E.

The cut diagram provides some information about the geometry of the polygon. For example, the chords in the cut diagram provide the points on the boundary of the polygon which are visible to each other. Therefore, the cut diagram is closely related to the notion of the visibility graph. It is yet an open problem to determine the geometry of a polygon corresponding to a given visibility graph. We conjecture that for cut diagrams this problem is easier, given that relaxing the need for straight edges in the environment boundary, it becomes trivial. However, we could not find a satisfying solution yet. As an example, the three different polygons having the same cut diagram are shown on Figure 3.

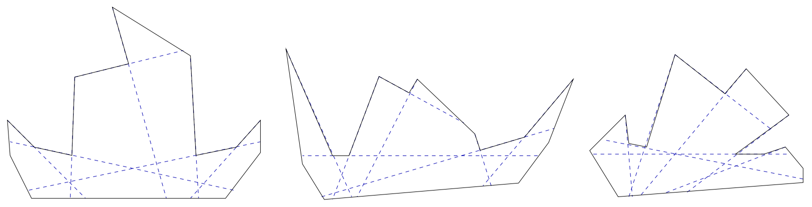

Fig. 3. Some polygons having the same cut diagram, shown on Figure 1.b.

3.1 Cut diagram properties

From the cut diagram, the Bitbot can infer visibility information. Such information can be recovered by recognizing a few properties of the cut diagram. The first property relates intersections of the cuts in the polygon with intersections of chords in the diagram:

Proposition 1. *Draw a segment between each pair of mutually visible points on the boundary of a polygon. These segments intersect pairwise in the polygon if the corresponding chords intersect pairwise in the cut diagram.*

Proof: When two segments, $\overline{x_1x_2}$ and $\overline{y_1y_2}$, intersect in the polygon, then the ordering of the corresponding endpoints along the boundary of the polygon is determined. For the closed curve that is going through all of the four endpoints without intersecting the segments and self-intersections, the ordering must be either (x_1, y_1, x_2, y_2), or (x_1, y_2, x_2, y_1). Since the bijection function preserves the ordering of the endpoints on the boundaries of both the polygon and the closed curve, the corresponding chords always intersect in the cut diagram. ∎

Corollary 1. *Two cuts intersect in the polygon if and only if they intersect in the cut diagram.*

Therefore, the cut diagram preserves the two-intersections of the cuts. Unfortunately, this is not the case with three-intersections. That is, if two intersecting chords are intersected by a third chord, the order in which the cuts intersect in the polygon may be different from the order in which the corresponding chords intersect in the cut diagram (see Figure 4).

3.2 Bitangents in a Cut Diagram

While there is no direct information about the location of bitangents and and their complements in the cut diagram, there is a necessary condition for existence of a bitangent between two vertices. Such condition is exploited for detecting possible bitangents in the cut diagram by the Bitbot. This becomes the base for the pursuit-evasion algorithm presented in Section 5. This important condition is expressed in the following proposition.

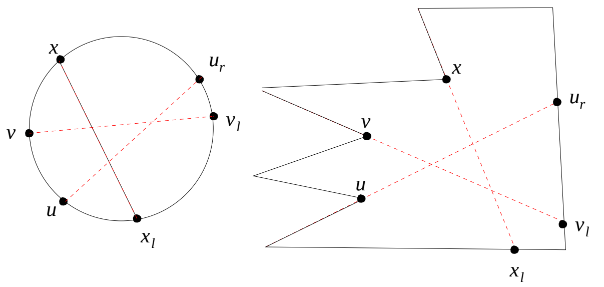

Fig. 4. The cut diagram does not always preserve three-intersections of the cuts. The order in which the cuts $\overline{xx_l}$, $\overline{uu_r}$, and $\overline{vv_l}$ intersect in the polygon is different from the order in which the corresponding chords intersect in the cut diagram.

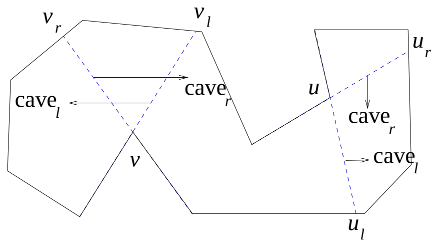

Fig. 5. The reflex vertex v belongs only to the cave$_r$ of the reflex vertex u and not to the cave$_l$. Similarly, u lies only in cave$_r$ and not in cave$_l$ of vertex v. Therefore, the necessary condition for bitangent existence between vertices u and v is satisfied. However, there is no bitangent between the two vertices.

Proposition 2. *If a bitangent exists between two reflex vertices, then each of the reflex vertices is inside exactly one of the caves of the other vertex.*

Proof: If any of the vertices is in both of the caves, then the extension of the bitangent will lie outside the polygon. ∎

Unfortunately, this is a necessary, but not a sufficient condition for the existence of a bitangent. Figure 5 shows a polygon in which a bitangent does not exist between the two reflex vertices u and v, even though the above condition is satisfied.

Consider now the set of all of the pairs of the reflex vertices in the polygon. Those pairs satisfying the previous condition are called *(bitangent) candidate pairs*. The set of candidate pairs includes the bitangent pairs as a subset. Therefore, using the above condition, it is possible to find the candidate pairs given a cut diagram, but it is not possible to discriminate the bitangent pairs. Figure 6 shows all of the possible configurations of the cuts in the cut diagram which produce candidate pairs.

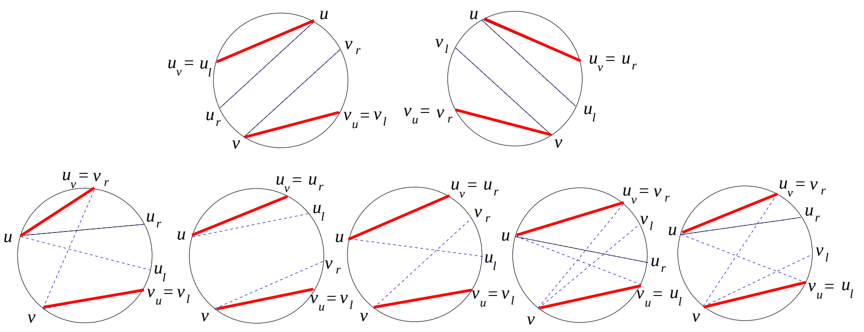

Fig. 6. All of the possible configurations of the cuts (dashed blue lines) of the two reflex vertices in an environment resulting in a candidate pair. In case the two reflex vertices are the only reflex vertices in the polygon, the corresponding approximations of the bitangent complements are also shown in bold (red).

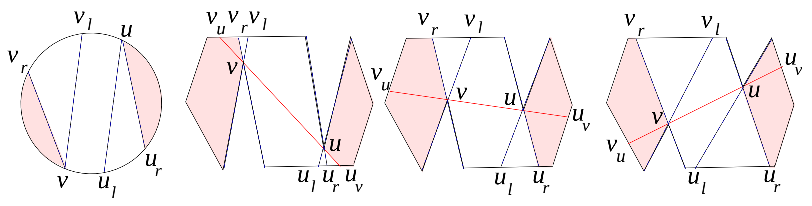

Fig. 7. The cut diagram has only two reflex vertices in the environment, forming the bitangent. Some of such cases are shown, together with the corresponding bitangent complements.

4 Inferring the Environment Geometry

This section is dedicated to the information about the bitangent complements that can be inferred from the cut diagram. Proposition 3 provides a first approximation on the location of the bitangent complements. In some cases the estimate can be further improved with some other properties of the cut diagram. To introduce these properties, consider a polygon with reflex vertices, u, and v, and their corresponding cuts' ends, u_r, u_l, v_r, and v_l. Throughout all of this section, assume that v belongs to the cave of the cut $\overline{uu_r}$. This is always true up to symmetry. Also, assume that u belongs to the cave of the cut $\overline{vv_q}$, in which q is either l or r. Thus, the two vertices form a candidate pair by Proposition 2.

Proposition 3. *If a bitangent exists between vertices u, and v, then the bitangent complement $\overline{uu_v}$ in the cut diagram is the chord that connects u with a point inside the circular arc $\widehat{uu_r}$.*

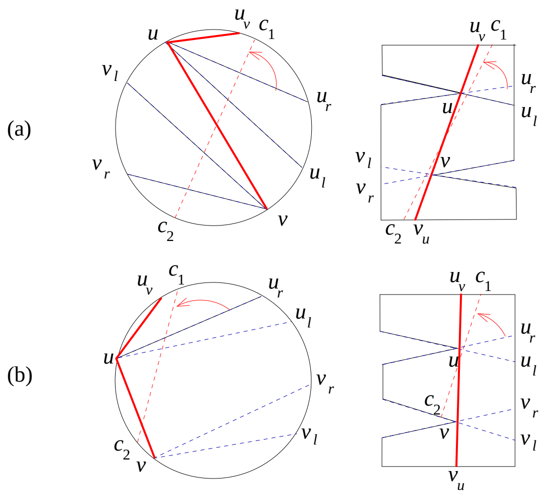

Fig. 8. If a straight line intersects the chord corresponding to the bitangent, then the bitangent complement must intersect a reduced portion of the arc $\widehat{uu_r}$.

Proof: Since the bitangent $\overline{uv}$ intersects the straight line containing the segment $\overline{uu_r}$ at point u, the bitangent complement $\overline{uu_v}$ cannot intersect the segment $\overline{uu_r}$ in any other point but u, since two lines intersect only at one point. ∎

Figure 7 shows an example of different instances of the polygon with the same cut diagram, with a bitangent complement connecting u with different points on the arc $\widehat{uu_r}$. Based on the principle, that straight lines must intersect properly in the polygon if they intersect in the cut diagram, we can reduce the bound on the places where a bitangent complement lies even to smaller regions, if other cuts are present in the cut diagram (Figure 8). This is described in the following proposition:

Proposition 4. *Let a cut $\overline{c_1c_2}$ intersect the chord $\overline{uv}$ in the cut diagram, such that $c_1 \in \widehat{uu_r}$. If a bitangent exists between u, and v, then the bitangent complement $\overline{uu_v}$ intersects the arc $\widehat{uc_1}$.*

Proof: If the bitangent intersects a line of the segment $\overline{c_1c_2}$, then the bitangent complement $\overline{uu_v}$ cannot intersect the segment $\overline{c_1c_2}$, because two lines cannot intersect in two points (see Figure 8). ∎

Definition 2. *Consider all of the cuts $\overline{c_1^ic_2^i}, i = 1..p$, such that each of them satisfies the condition of Proposition 4. Consider corresponding arcs $\widehat{uc_1^i}$. Pick an index k, such that $\widehat{uc_1^k}$ is the smallest arc, $\widehat{uc_1^k} \subset \widehat{uc_1^i}, \forall i$. The segment $\overline{uc_1^k}$ is called an approximation of the bitangent complement $\overline{uu_v}$.*

Proposition 4 provides a bound on the range where the bitangent complement may intersect the boundary of the polygon. We conjecture that this bound is always tight for a cut diagram.

Conjecture 1. If a bitangent exists between u, and v, then, for any $\epsilon > 0$, there exists a polygon E, in which an endpoint u_v of the bitangent complement $\overline{uu_v}$ is arbitrarily close to the endpoint c_1^k of its approximation $\overline{uc_1^k}$, with the distance between them $|\overline{u_v c_1^k}| < \epsilon$.

Proposition 5. *Let u and v be the only two vertices forming a bitangent candidate pair in the cut diagram. There always exists a polygon E such that the endpoint u_v is arbitrarily close to the endpoint u_r.*

Proof: To construct such a polygon, the angle between the lines which contain the bitangent $\overline{uv}$ and the inflection $\overline{uu_r}$ have to approach zero, as shown on Figure 7. ∎

If two vertices form a candidate pair, but they do not form a bitangent in the polygon, then the two vertices are not mutually visible. In this case an approximation of a bitangent complement may not be meaningful for the given polygon. However, the following proposition is helpful for analyzing this case:

Proposition 6. *If there is no bitangent between the bitangent candidate pair u and v, then there exist reflex vertices w, z, such that both pairs u, and w, and v, and z are bitangent pairs (the vertices w and z may be the same vertex).*

Proof: If u and v are not mutually visible, then its shortest path in E contains at least one reflex vertex. Call this vertex y. The vertex y always exists, since, if all of the vertices of the path were convex, then $\overline{uv}$ would be inside the polygon, which is a contradiction. If y is visible from u, then assign y to w, and the bitangent pair for u has been found. If y is not visible from u, the previous process is repeated, until a reflex vertex is found, which is visible from u. This process has to stop, since there is a finite number of vertices. With a similar analysis the vertex z can be found. ∎

The intuition behind Proposition 6 is the following. If a candidate pair does not form a bitangent pair, one of the vertices is 'hidden' behind a real bitangent complement of another vertex. While the Bitbot cannot discriminate between candidates and real bitangents, Proposition 6 allows to make some inferences about the combinatorial changes of the shadow region. This is explained in the next section, when the pursuit-evasion algorithm is described.

5 Pursuit-Evasion with Bitbots

In this section we present a pursuit-evasion algorithm using the cut diagram described in previous sections. Note, that without any additional sensor for detecting an evader, the Bitbot can never “detect” it in the usual sense. To resolve this, we provide the Bitbot with a sensor for detecting the presence (though not the location) of targets. Equivalently, we could consider a slightly modified pursuit-evasion game, in which the evaders are moving unpredictably, but are willing to be found (for example, lost people in the building on fire). When they (the evaders) see the Bitbot, they are considered to be detected. Both of these approaches lead to the same strategy and provide the ground for considering pursuit-evasion tasks for Bitbots. To make discussion simpler, we consider the pursuit-evasion game in which an evader is considered to be detected as soon as it comes in the line of sight of the pursuer.

5.1 Critical Events in the Cut Diagram

The visibility-based pursuit-evasion problem can be solved by keeping track of all of the shadow components in the environment, since they are the only places where the evaders can hide [7]. This is done by labeling each shadow component as *contaminated*, if an evader may be hiding behind the gap, or *cleared*, otherwise. As the robot moves, the labels of the regions change according to certain critical events, and the goal of the algorithm is to find valid Bitbot movements such that all of the shadow regions are labeled as cleared. An *information state* for the pursuit-evasion task is defined as the cut diagram, together with the labeling of the cuts and the position of the Bitbot with respect to the cut diagram. The goal of the algorithm is to reach an information state in which all of the cuts are labeled as cleared. The information state is defined as the space of all of the possible labelings for a cut diagram of a polygon, with all of the possible positions of the Bitbot in the cut diagram. As the Bitbot moves, the information state changes according to the following critical events:

- Crossing an inflection ray of the polygon. After this event either a new shadow component appears or an old shadow component disappears.
- Crossing a bitangent complement of the polygon. In this case either two shadow components merge, or split.

For the Bitbot, crossing an inflection ray corresponds to crossing a cut, which can be determined in the cut diagram. The second event can be *conservatively* detected, by crossing the approximates of bitangent complements of the bitangent candidate pairs. If a shadow component appears, it is labeled as cleared. If one contaminated component merges with a cleared one, the new shadow component is labeled as contaminated. When a shadow component splits, the new shadow component inherits the labeling of the component that splitted. Each shadow component is associated with the corresponding cut and cave of the environment (see Figure 9).

A cut of a reflex vertex u is labeled as contaminated if the corresponding cave may contain an evader in the shadow component associated with u.

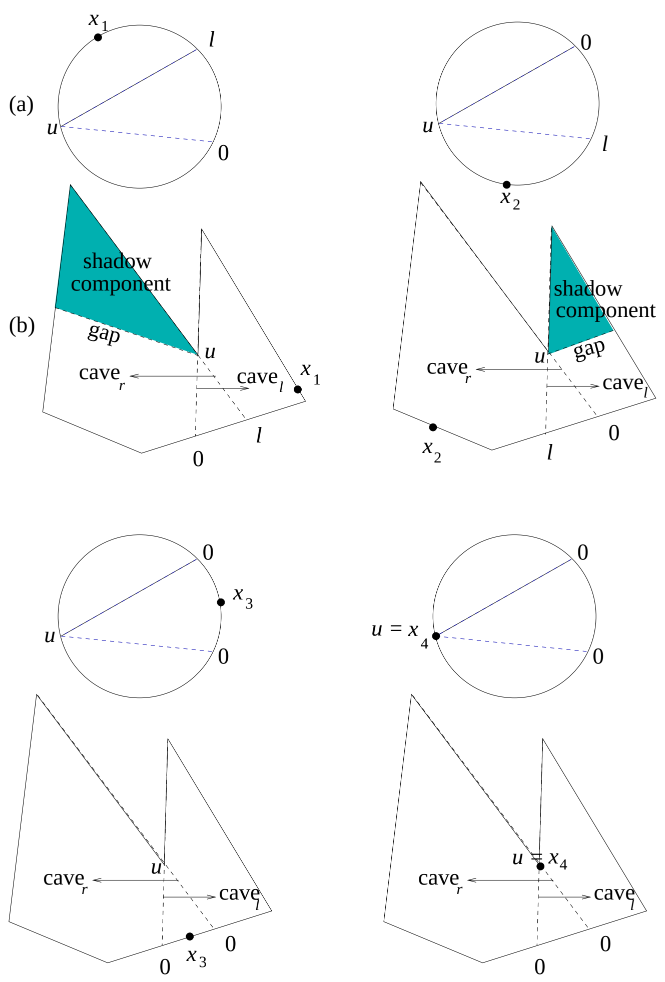

Fig. 9. For a reflex vertex u the two cuts and corresponding caves are shown. For positions x_1, and x_2 of the Bitbot, the gap associated with u and corresponding shadow component may be only inside one of the caves (the one that does not contain x_1 or x_2). If the shadow component exists, the cut of its cave has the label, l, corresponding to whether the shadow component is cleared or not. The other cut has label cleared. For positions x_3, and x_4, the shadow component associated with the vertex u does not exist, therefore, the cut labels are cleared. The corresponding configurations on the cut diagram are also shown.

Otherwise, if the shadow component of u cannot contain an evader, the cut is labeled as cleared.

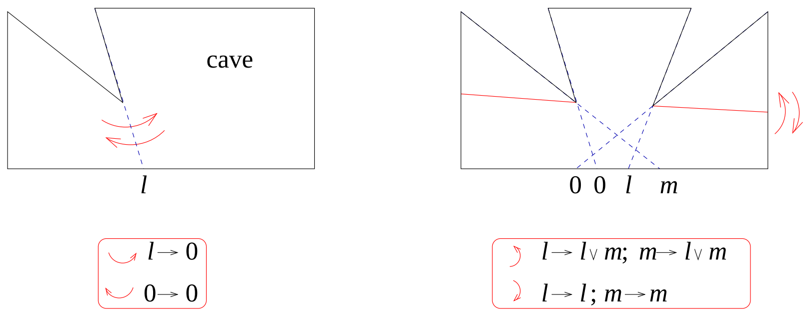

Fig. 10. Critical cut events and relabeling of the cuts.

The Bitbot clears a cut by entering the corresponding cave. This does not imply that the whole cave is cleared. This would only happens when each of the shadow components inside of the corresponding cave was cleared. Thus, at least one of the cuts for each reflex vertex has a label cleared. Note, that for certain locations both of the caves are labeled as cleared (see Figure 9, positions x_3 and x_4). Initially, when the Bitbot has not searched the environment yet, the labeling of the cuts marks all of the cuts containing shadow components as contaminated and others as cleared, according to the patterns shown on Figure 9.

The cuts are used to conservatively infer the shadow critical events and labelings. Here, by conservatively we mean that if a critical event exists in the polygon, it is detected through the cuts, but if there are events detected through the cuts these events do not necessarily exist in the polygon. When the Bitbot crosses a cut (Figure 10.a), it can be either entering the cave corresponding to the cut or leaving it. When it enters a cave, the shadow component that has been in that cave disappears. Therefore, the label of the cut changes to cleared. When the Bitbot leaves a cave, the shadow component appears in this cave. This shadow component has just been visible to the Bitbot, therefore, the label of the cut remains cleared. When a Bitbot crosses a bitangent complement (Figure 10.b) and several shadow components merge into one, the labels of corresponding cuts should all be set to contaminated, if there was at least one contaminated cut among the merging shadow components. When a shadow component splits into several, all of them inherit the same labels they had before (which is the label of the splitting shadow component).

A first difficulty is that the Bitbot does not know the exact location of the bitangent complements, which are needed to compute the merges and splits of the shadow components. To overcome this, the approximations described in Section 4, are used. Another difficulty appears when two vertices in the candidate pair do not form a bitangent pair. Crossing the corresponding bitangent complements of these vertices will result in relabeling of the cuts, when merging of shadow regions does not happen. We prove that in this case the relabeling that occurs is consistent with the correct labeling of the cuts. Consider two reflex vertices forming a candidate pair u, and v. Let all of the assumptions made in Section 4 hold. Imagine that the bitangent between the vertices does not exist. Based on Proposition 6 there exists a vertex w which forms a candidate bitangent pair with both of u and v.

Proposition 7. *If the bitangent complement of the pair u and w is $\overline{ux_1}$, and a bitangent complement of the pair u and v is $\overline{ux_2}$, then $\widehat{ux_2} \subset \widehat{ux_1}$.*

Proof: Suppose that the contrary is true, that is $\widehat{ux_1} \subset \widehat{ux_2}$. Then there must exist a chord $\overline{xx_1}$ that crosses the chord $\overline{uw}$ such that $x_1 \in \widehat{ux_2} \subset \widehat{uu_r}$ by definition of the approximation of the bitangent complements. Such chord also crosses $\overline{uv}$. Therefore, the approximation of the bitangent complement of u and v must be $\overline{ux_1}$, which contradicts the assumption. ∎

This means that when the shadow component of v contaminates the shadow component of u, even though none of the critical events happen in the polygon, the shadow component of u has already been contaminated by w, which originally contaminated v.

Note, that the Bitbot cannot process events as they appear. This is because the three-intersections of the cuts are not preserved in the cut diagram. An example of several cut events happening while the Bitbot moves from state x_1 to state x_2 is shown on Figure 11. When a cut event occurs, only the label of this cut can be changed. Therefore, the cut events are independent from each other, and the Bitbot can process all of them in state x_2, changing the labels according to the rules of cut events. Processing labels when crossing a bitangent complement requires more consideration. The events may be dependent (many cuts can be relabeled when one recontamination happens) and the order in which they occur is not known. However, since the approximations of bitangent complements of the same vertex intersect in that vertex, they cannot intersect inside the cut diagram. Therefore, the three-intersections can occur only between the approximations of the bitangent complements of three different vertices. Therefore, these events are also independent. An example of multiple critical events, occurring when the Bitbot moves from state x_1 to state x_2, and then to state x_3, is shown on Figure 12.

5.2 The Algorithm

The algorithm performs a breadth-first search on the information space of all of the cut diagrams with cut labelings. From the starting information state (the cut diagram with initial labeling) all of the possible Bitbot moves are tried, relabeling using the rules described in Section 5 is performed, resulting information state is computed and checked with the goal state (the cut diagram with all of the cuts cleared). The same procedure is iterated over all of the resulting information states.

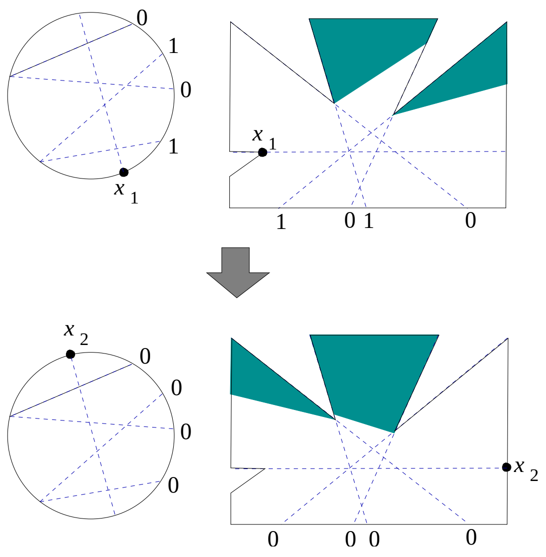

Fig. 11. An example of multiple cut critical events, when the Bitbot moves from the vertex x_1 to x_2. All the shadow components become cleared.

The completeness of the algorithm depends on the quality of approximations used for bitangent complements in the cut diagram.

Proposition 8. *If Conjecture 1 is true, then the algorithm presented above is Bitbot-complete; a Bitbot will find a solution if one exists.*

Proof: When the Bitbot relabels the cuts as recontaminated, there exists a corresponding environment, in which the actual recontamination happens, by Conjecture 1. Therefore, the algorithm presented above would be complete in the sense that, if there exists a solution for all of the environments corresponding to the cut diagram, one will be reported. ∎

6 Implementation and Experimental Results

We have implemented a simulation of the algorithm presented in Section 5.2. We present experiments with three different environments. The first experiment is shown on Figure 13. We show the cut diagram with all of the cuts and approximations of the bitangent complements. Using only the cut diagram and the algorithm for pursuit-evasion, the Bitbot generates the plan for finding all of the evaders in the environment. The produced solution path is shown. In the experiments shown on Figure 15 the pursuit-evasion algorithm was used to produce the clearing paths for the other two environments.

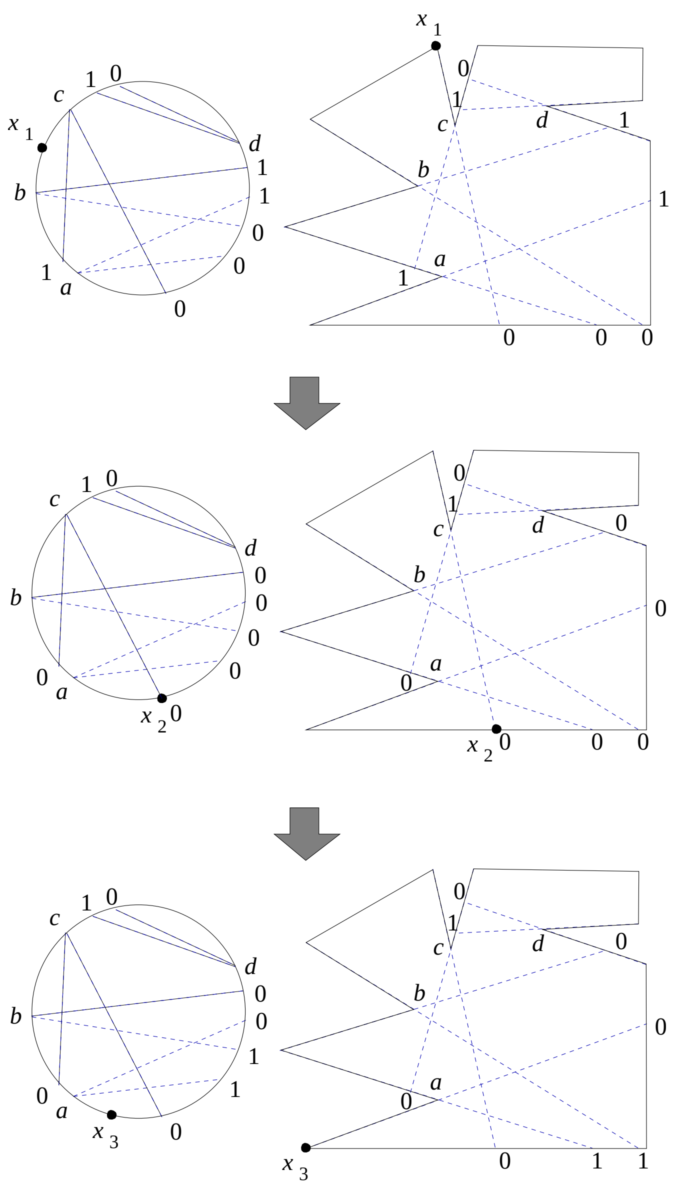

Fig. 12. An example of multiple critical events, when the Bitbot moves from the state x_1 to x_2 and then to the state x_3.

7 Conclusions

In this paper we introduced very simple robots called Bitbots. The sensor and control limitations of Bitbots allow careful modeling of the information space

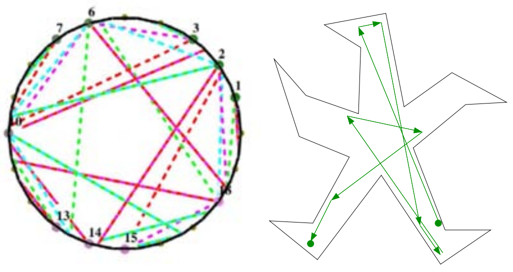

Fig. 13. For the given environment, the cut diagram with all of the cuts and the approximations of the bitangent complements, generated by our program, is shown on the left. The solution path for the pursuit-evasion algorithm is shown on the right.

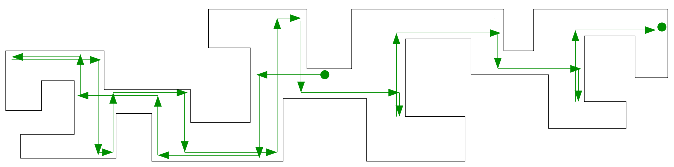

Fig. 14. Computed solution paths for catching the evaders in the given environments.

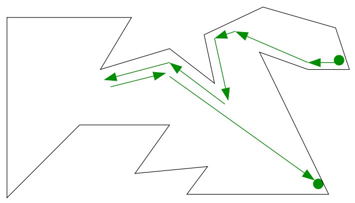

Fig. 15. Computed solution paths for catching the evaders in the given environments.

arising when presenting visibility tasks to these robots. We have formally defined the tasks of pursuit-evasion as planning problem in corresponding information space and presented solution and simulation results.

The notion of information spaces, which is the main theme of this paper, is fundamental to robotics. However, little or nothing is known about many aspects of information spaces. Almost no work has been done on characterizing their topology or geometry. No general techniques are available for planning in information spaces. Our goal is to continue researching information spaces arising in robotics.

References

1. S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Sensorless parts feeding with a one joint robot. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 229–237. A K Peters, Wellesley, MA, 1997.
2. G. Dudek and C. Zhang. Vision-based robot localization without explicit object models. In *IEEE Int. Conf. Robot. & Autom.*, pages 76–82, 1996.
3. M. Erdmann, M. T. Mason, and Jr. G. Vaněček. Mechanical parts orienting: The case of a polyedron on a table. *Algorithmica*, 10:206–247, 1993.
4. M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Trans. Robot. & Autom.*, 4(4):369–379, August 1988.
5. K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.
6. K. Y. Goldberg and M. T. Mason. Bayesian grasping. In *IEEE Int. Conf. Robot. & Autom.*, 1990.
7. L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999.
8. T. Kameda, M. Yamashita, and I. Suzuki. On-line polygon search by a six-state boundary 1-searcher. Technical Report CMPT-TR 2003-07, School of Computing Science, SFU, 2003.
9. S. M. LaValle. *Planning Algorithms.* [Online], 2004. Available at http://msl.cs.uiuc.edu/planning/.
10. J. M. O'Kane and S. M. LaValle. Almost-sensorless localization. In *IEEE Int. Conf. Robot. & Autom.*, 2005.
11. J. M. O'Kane and S. M. LaValle. Global localization using odometry. In *IEEE Int. Conf. Robot. & Autom.*, 2006.
12. J. O'Rourke. Visibility. In J. E. Goodman and J. O'Rourke, editors, *Discrete and Computational Geometry*, pages 467–481. CRC Press, New York, 1997.
13. B. Tovar, L. Guilamo, and S. M. LaValle. Gap navigation trees: Minimal representation for visibility-based tasks. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2004.
14. A. Yershova, B. Tovar, R. Ghrist, and S. M. LaValle. Bitbots: Simple robots solving complex tasks. In *AAAI National Conference On Artificial Intelligence*, 2005.