Gap Navigation Trees: Minimal Representation for Visibility-based Tasks

Benjamín Tovar, Luis Guilamo, and Steven M. LaValle

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

Abstract. In this paper we present our advances in a data structure, the Gap Navigation Tree (GNT), useful for solving different visibility-based robotic tasks in unknown planar environments. We present its use for optimal robot navigation in simply-connected environments, locally optimal navigation in multiply-connected environments, pursuit-evasion, and robot localization. The guiding philosophy of this work is to avoid traditional problems such as complete map building and exact localization by constructing a minimal representation based entirely on critical events in online sensor measurements made by the robot. The data structure is introduced from an information space perspective, in which the information used among the different visibility-based tasks is essentially the same, and it is up to the robot strategy to use it accordingly for the completion of the particular task. This is done through a simple sensor abstraction that reports the discontinuities in depth information of the environment from the robot's perspective (gaps), and without any kind of geometric measurements. The GNT framework was successfully implemented on a real robot platform.

1 Introduction

This paper centers on the development of mobile robotics systems that perform sophisticated visibility-based tasks with minimal sensing requirements. The goal is to provide autonomous mobile robots for applications such as surveillance, search-and-rescue, firefighting, law enforcement, and remote visual presence. Classical approaches often lack reliability when applied in practice due to problems such as mapping uncertainty, registration, segmentation, localization errors, and unpredictable control errors. A primary cause is that previous algorithmic efforts have often assumed the availability of perfect geometric models. The guiding philosophy in this work is to avoid most of these difficulties by developing algorithms and mobile robots that minimize the information requirements. By constructing an algorithm and control law that use the information directly from the robot sensors, it may be possible to complete a task while eliminating the need to make potentially flawed measurements. This can provide low-cost solutions to challenging problems, while achieving greater reliability in the face of uncertainties.

In this work we present our developments of a dynamic data structure called the *Gap Navigation Tree* (GNT). The GNT is constructed entirely from online sensor measurements made by the robot. Once constructed, it encodes paths from the current position of the robot to any place in the environment. As the robot moves, the GNT is updated as to always reflect the path information from the current position of the robot. These paths are globally optimal in distance if the environment is simply-connected, even though geometric information, such as lengths, angle measurements and robot orientation, is not available. The only restriction is that the components of the environment's boundary must be piecewise smooth closed curves, with only a finite number of nonsmooth points. The approach is based on modeling an abstract sensor that reports the order of depth discontinuities with respect to the heading of the robot. These discontinuities are called *gaps*. Encoding the changes of the gaps, the information state of the robot evolves, until a goal state is reached. Such goal state is task dependent, and it may be, for example, to explore the whole environment, to find all evaders in pursuit tasks, etc. The gap changes, called gap critical events, correspond to combinatorial changes in the visibility region of the robot. Given that no exact geometric information is needed for its construction or use, the GNT provides a robust framework for solving robotic tasks in the plane. Moreover, it provides a simple sensor-feedback strategy if the path that the robot transverses is encoded in the GNT.

The paper is organized as follows. The remainder of this section is a brief review of the previous work related to our approach. Section 2 presents the sensing model, followed by a description of the GNT in an information space framework in Section 3. Section 4 sketches the use of the GNT in robot localization, navigation and pursuit-evasion tasks.

1.1 Sensorless and minimal sensing settings

In this paper we are concerned with visibility-based tasks in the plane. More specifically, those robotics tasks for which sensor information can be modeled as a visibility region. The visibility region is the set of points that can be joined with a line segment to the robot's position, without intersecting the environment's boundary. As the robot moves, its visibility region changes, modifying the information that the robot knows about the environment or the progress towards a goal. We are interested in determining the minimal sensing capabilities a robot should have in order to complete a visibility-based task. One approach that considers minimal sensing settings is known as *bug algorithms* [16,20]. Combining global knowledge with local information, a robot is able to navigate among obstacles and reach a given goal. The robot navigation capabilities are simple (movement towards obstacles and wall-following), no representation of the environment is kept, and the global information may consist only of the position of the goal to reach. These

characteristics allow the use of bug algorithms in robots with very limited sensing capabilities and unreliable motion control.

In general, the initial state of the robot is unknown, and as it moves in the environment, it might reach a goal state without determining its exact current state. In robotics, the problem of driving a system from an unknown state to a goal state was first considered in the context of manipulation [10]. For example, up to convex hull symmetry it is possible to manipulate polygonal parts to a final configuration without any sensor information [11]. Of course, not all robotic tasks can be solved without sensors, but it is very interesting, and scientifically relevant, to determine the minimum information necessary to complete a given task [2,6]. Moreover, one may go a step further and design a sensor that exactly suits the robotic task. One can think of an *abstract sensor* that gives the "ideal" minimal information to the robotic system to work correctly, and its physical implementation in a "subideal" sensor [9]. This philosophy can be used for solving the classical path-planning problem, as presented in [28]. Using a critical-point detector and a passage-point detector as abstract sensors, collision avoidance is achieved with sensor-based repulsive potentials.

1.2 Visibility-based tasks

The changes in the visibility region have been extensively studied, from the art-gallery problem [23], to decompositions of the environment into regions of *similar* visibility. In [26], a cell complex decomposition is presented, the *visibility complex*, in which points inside a cell *see* the same set of objects in the environment. The space can be decomposed also into equivalence classes of similar visibility of an object. Elements inside a class have a similar qualitative view of the object, that is, see the same *aspect*. An aspect here may be defined as the set of views of an object that share the same combinatorial structure. This leads to the study of the *aspect graph* [4]. In [13], a planar environment is decomposed into cells that see the same aspect of the environment's boundary. The combinatorial structure of visibility regions in that work is defined in terms of *spurious* and *non-spurious* edges. A spurious edge is an edge of the visibility region that does not exist in the environment's boundary, but it is produced by occlusions (i.e., by a corner). The supporting line of a spurious edge is always collinear with the robot's position. The sequence of spurious and nonspurious edges define the *visibility skeletons*, and the environment is decomposed into cells of points that share the same visibility skeleton. Such decomposition is called the *visibility cell decomposition*.

In the decompositions mentioned before, if the robot moves inside a cell there is not significant change in information. The robot receives about the same information from the sensors. Such movements are called *conservative* in the sense that they preserve the current robot's information. In contrast, when the robot crosses one of the cells' boundary edges, the structure of the

visibility region suffers a drastic change, and the robot's information may be modified. Such sudden changes are called *visual events*[8].

In the plane, if the components of the environment's boundary are represented as a regular piecewise smooth closed curves, there are two kinds of visual events. One kind is triggered when the robot crosses an environment's boundary generalized inflection ray, and the other when it crosses the complement of bitangent line segments of the boundary[1]. An *inflection* is a change in the sign of the curvature of the environment's boundary. We use the term "generalized," as in [19], to include polygonal boundaries. Given a generalized inflection, an *inflection ray* is found by extending a ray from the inflection until it hits another point of the environment's boundary. A bitangent line segment is a segment completely contained in the environment representation, whose supporting line is tangent to two points of the boundary, and whose endpoints are these points of tangency. A common general position assumption is that no line is tangent to more than two points of the boundary (thus the term *bitangent*). For each bitangent, its *complement* is found by extending outward from each point of tangency until the environment's boundary is hit again (see Figure 2).

Since the inflection and bitangents are so relevant for visibility-based problems, one may wonder if the robotic tasks may be described *solely* in terms of inflections and bitangents, forgetting all the rest of geometric information of the visibility region. Visibility-based tasks may have then the form of states of information that are modified each time a visual event is detected. This relation between the visual events and the robotic tasks has been exploded before for different visibility-based tasks. Our aim here in the rest of this Section is to give an unified view of visibility-based tasks. Particularly, we will try to convince the reader that the tasks of optimal navigation, robot localization, and pursuit-evasion share the same information structure. It is up to the robot's strategy to choose the way of manipulating this structure to solve the particular task.

The first task we describe is optimal navigation in the plane. The classical algorithm is to perform a search in the visibility graph, or in the bitangent graph [22]. The bitangent graph is obtained from the visibility graph by deleting all the edges, that if grew by a small epsilon, at least one of the endpoints lies outside the boundary, or inside an object[2]. The bitangent graph is entirely defined in terms of inflections and bitangents. Taking advantage of this, in [1], shortest path trees are used to efficiently update the visibility region of a moving point. On the other hand, each time a visual event occurs, a portion of the shortest-path structure is revealed to the robot. It then becomes

[1] In polygons there is a third kind of visual event, in which a non-reflex vertex comes into view. Although this is relevant for defining, for example, the visibility skeletons, it is particular for polygons, making generalization to curved environments difficult.

[2] Thus, the bitangent graph is called the *reduced visibility graph* in [18].

possible to retrieve the shortest-path trees of an unknown environment, as it is presented in [31,32]. Note, however, that the task of getting the shortest-path information may use longer paths itself. This is because, in general, for unknown environments, the competitive ratio between the length of the path generated by any navigation strategy and the length of the optimal path is unbounded [24].

In the robot localization problem, the robot must infer its position based on local sensing [3,5,13,17]. One approach is to generate position hypotheses by comparing the visibility skeleton of the robot's visibility region with the skeletons of the visibility cell decomposition[13]. If there is more than one position hypotheses, the robot should move around to determine its position with certainty. In [7], a strategy that computes an overlay arrangement of the visibility cells is used to determine the optimal movement strategy to reduce the set of position hypotheses. In that approach, a decision tree is simulated, in which each decision node is a visibility skeleton comparison that discards some position hypotheses. From the robot perspective, the localization problem can be stated as series of visual events that (must) occur to uniquely identify the robot's position. For example, if the robot detects two bitangents and one inflection, in that order, it can discard all the possible positions where the same movements lead to three bitangents in a row. This sequence of visual events is available from the shortest-path tree[1]. Using only visual events (i.e., no compass) will localize the robot up to a rigid rotation transformation of the environment. In fact, the assumption of a compass has a big impact in robotic tasks [2,6].

The last visibility-based task we describe here is the pursuit-evasion problem. In this task, one or several *pursuers* should find all possible *evaders* that may be hiding in the environment[30]. In this task, a region of the environment is said to be *contaminated* if it might contain an evader, otherwise it is referred as *cleared*. A cleared region that is contaminated again is called *recontaminated*. The pursuit-evasion problem is solved when all the regions of environment are cleared. Cleared and contaminated regions are separated from the robot's perspective by the current visibility region. Cleared regions may only get recontaminated if they *merge* with a contaminated one, which can only be the result of a combinatorial change in the visibility region. These combinatorial changes are the base of the search algorithm presented in [12]. In terms of visual events, an strategy should dictate the sequence of inflection rays and bitangent complements to cross in order to clear the whole environment. In [29] a strategy is presented that explicitly avoids some visual events of being triggered to clear an unknown environment. The order of how the visual events should occur also dictates the structure of the "T", "S" and "Z" patterns in [15], in which a 6-state automaton restricted to move along the environment boundary serves as a pursuer. In [14] an algorithm is presented in which clearing the environment is equivalent to clearing the shortest-path structure from the current robot position in an unknown environment.

2 Sensing model

In this Section we formalize the model of the robot and the model of the information used from the sensors. The robot is modeled as a moving point in a connected open set R in the plane. Let $O = \{o_1, o_2, ..., o_n\}$ be the possibly empty set of pairwise disjoint obstacles, in which $o_i \subset R$ for $i = [1, 2, \ldots, n]$ is a closed set. Let ∂o_i be the boundary of $o_i \in O$. We assume that ∂o_i has a single connected component. Let $F = R \setminus \bigcup_{i=1}^{n} o_i$, with $o_i \in O$, be the free space. Let ∂F be the boundary of F. The components of ∂F are piecewise smooth closed curves with only a finite number of nonsmooth points. The robot is only able to move in F. In Section 5 we will briefly discuss the case when the robot is not a point.

The problem here is to design a sensor that detects the combinatorial changes in the visibility region. As we will detail later, the changes of the visibility region can be defined as a function of the spurious edges. When a spurious edge either appears, disappears, splits or merges with another, a combinatorial change in the visibility region occurs. From the robot perspective, the spurious edges are the discontinuities in depth information in the environment. Note that geometric information of the spurious edges is not relevant for the visual event detection. The events will be the same in spite of the exact length and angular position of the spurious edges. Their order is relevant, though, since we are interested in which discontinuity disappeared, or merged with another, for example. Although the precise distances to the walls may be unknown, the robot only needs a type of edge detector that can detect each of the discontinuities, and return their direction relative to the robot's heading. From now on, each of this discontinuities will be referred to as a *gap*, and the sensor as a *gap sensor* [27]. Besides the gap sensor, no other sensing ability is assumed, i.e., it has neither a compass nor a reliable odometer. The ideal gap sensor can be easily realized with through a range sensor (i.e., laser or sonar) or using computer vision techniques.

Each gap *hides* a connected region of the environment that is occluded to the robot from its current position. A label of "L" or "R" is assigned to a gap to indicate the direction of the part of R that is hidden behind the gap. This corresponds to transitions of the gap sensor from "far to near" (left) or "near to far" (right), if the gaps are detected by a counterclockwise scan with respect to the robot's heading (see Figure 1.(a)).

2.1 Gap Tracking

When the robot moves in the environment, the gaps, as reported by the gap sensor, may change. We refer to *gap tracking* to the process by which the gap sensor monitors the current visible gaps and their respective changes. It is assumed that the robot can track the gaps at all times and record any topological change. There are four possible ways in which gaps change:

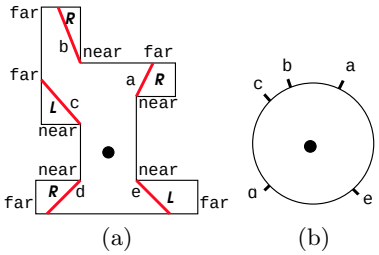


Fig. 1. The robot's view of the environment. The position of the robot is shown with a black disk. (a) The environment and the respective labeling of the gaps detected. (b) Angular position of the gaps detected in the visibility region.

Gap appearance: A gap, not detected before, is now tracked by the gap sensor. The gap is said to be visible.

Gap disappearance: A gap is no longer detected by the gap sensor. The given gap is not visible for the gap sensor.

Gaps merge: Several gaps merge into a single one.

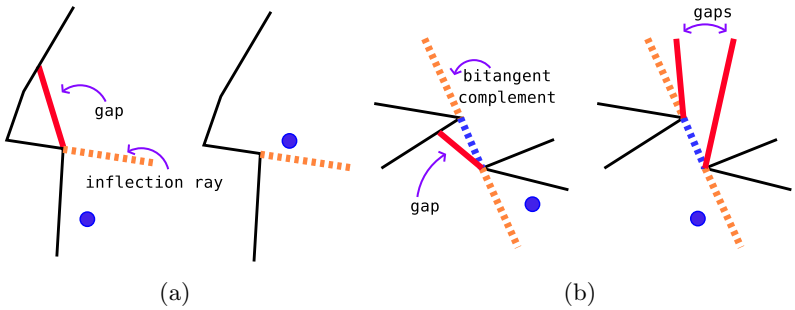
Gap split: One gap splits into several gaps.

If a gap appears, the region behind it was just visible to the robot, but now is "hidden" by the gap. Similarly, when a gap disappears, the region of the environment behind the gap is now visible to the robot. With bitangents, exactly two gaps may merge into one, and one gap splits exactly into two gaps. These four gap topological changes are called the *gap critical events*. We say 'critical events' instead of 'visual events' to make clear that the complete information of the visibility region is not available.

Appearances and disappearances of gaps are related to generalized inflections of ∂F. As illustrated in Figure 2.(a), appearances and disappearances of gaps occur when the robot crosses inflection rays. Merges and splits of gaps, are related to the bitangents of ∂F, and they occur when the robot crosses bitangent complements. (Figure 2.(b)).

With bitangents, exactly two gaps may merge into one, and one gap splits exactly into two gaps. With the minimal capabilities assumed now for the gap sensor, the robot cannot discriminate between a gap splitting in three, or a gap splitting into two, and an appearance close together. The general position assumption may be relaxed if a more sophisticated gap sensor is available (i.e., measuring exact distances or angles).

Given the gaps that the robot detects at a given time, it is possible to command the robot to move toward a given gap. This sensor-feedback movement is defined as *chasing a gap*. Note that with only the gap sensor information the only movements that are guaranteed to be collision free are gap chasing movements.



(a) (b)

Fig. 2. Inflections and bitangents of ∂F. (a) Appearance and disappearance of gaps occur when the robot crosses inflection rays. (b) Splits and merge occur by crossing bitangent complements.

3 The GNT from an information state perspective

Before introducing the GNT data structure, we define what we mean by an information space and an information state. As a state space captures the dynamics of a system, allowing the design of control inputs, or the study of the system trajectories, an *information space* captures the evolution of a system in terms of previous sensor readings and control inputs. Formally, let $X \subset \Re^n$ be the state space, $U \subset \Re^m$ the input space, and $u : [0, \infty) \rightarrow U$ the input history. Also, $Y \subset \Re^k$ is defined as the sensor space, which models the set of all of the possible instantaneous readings from the sensors. Let $y :\rightarrow [0, \infty) \rightarrow Y$ be the sensor history, in which $y(t) = h(x(t))$ is the instantaneous sensor reading for the state $x(t) \in X$ for some mapping $h : X \rightarrow Y$. An *information state* at time t is given by $\eta_t = (u_t, y_t)$, that is, the input and sensor history up to time t. The *information space* I is defined as the set of all possible information states. Given an information state η_t and an initial set of states $X_0 \subset X$, a *derived information state* $F(\eta_t, X_0)$ is the set of all the possible $x(t) \in X$ given η_t and X_0.

In our sensing model, the sensing space Y is defined by the set of all the of ordered circular sequences of possible readings of gaps. Thus, $\{L, L, R\} \in Y$ correspond to a sensor reading where two "left" gaps and a "right" gap are detected. Note this sensor reading is indistinguishable from $\{R, L, L\}$ and $\{L, R, L\}$, since a compass is not available. This means that the mapping h, from states to sensor readings is not injective. Even more, with only gaps readings, the exact position of the robot cannot be determined, and different neighborhoods of points will generate the same sensor reading across the whole environment. The input space is determined by the gap chasing movements, that is, the commands to the robot to move towards a gap, or a sequence of gaps.

3.1 Encoding information states

Remember that the robot can track the gaps all of the time and record any of their topological changes. Thus, it can detect that from the transition $\{L_1, R_1, R_2, L_2\}$ to $\{L_1, R_2, L_2\}$, the gap R_2 disappeared. The gap sensor only will report to the robot that a gap, detected before in this order, disappeared, for example. This identification of gaps is implicit at the sensor level, and it is possible if we assume coherency between the robot's motion and gap changes (i.e., small position changes of the robot will produce small angular position changes in the gaps).

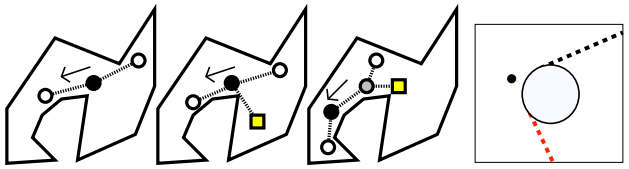
The gaps and their topological changes are encoded into a tree, hereafter referred to as T. The tree T is the *Gap Navigation Tree* of the environment. The root of T moves along with the robot. Each child of the root represents a gap that is currently visible, and they are maintained in the circular order of the gaps they represent. In T, we will use the terms gaps and nodes interchangeably because each node encodes a gap.

As the robot moves, critical events are triggered. As events occur, T is updated as follows: if a gap disappears, the corresponding node is removed from T. If a gap appears, it is added as a child of the root of T in a location that preserves the circular ordering of gaps. Any node that is added in this way is designated as a *primitive node*, which indicates that a portion of the environment that was once visible is now occluded. If a gap splits, then the corresponding child of the root will be replaced with two children. If two gaps merge, the two corresponding children of the root become the children of a new node, d, and d becomes a child of the root (see Figure 3.a). Merging can only occur between a pair of gaps that are adjacent in the circular ordering produced by the gap sensor. Each node also keeps the label associated with the gap.

A sequence of nodes from the root of T to a leaf define a sequence of gaps, that if chased, follows a path in the bitangent graph. The proof of this result is presented in Section 4.2. The relation of T with an information state is immediate. In fact, T is nothing more than the sensor history of the current information state. For example, assume that at time t_1 the state of T is T_1. At t_1 the robot is commanded to chase the sequence of gaps α, which brings T into the state T_2. Comparing T_1 and T_2 we can readily obtain the sequence α followed (input history), with the respective changes as reported by the gap sensor (sensor history). Note, however, that we are assuming that α is the shortest sequence of gaps to take T_1 into T_2. In this sense, all sequences of gaps that take T from one state to another are *equivalent* to the shortest one, since at the end, they modify T in the same way.

3.2 Building the GNT of an unknown simply-connected environment

We say that the robot is *exploring* if it is building the GNT of an environment. When the robot is placed in an unknown environment, all of the leaves of



(a) (b)

Fig. 3. (a) Encoding of critical events into a Gap Navigation Tree. The black disk denotes the root of T and the current position of the robot. As the robot chases the gap on the left, a gap appearance, and a gap merge are triggered. The structure of T is updated accordingly. (b) Absence of gap critical events in multiply connected environments. If the robot chases any of the to gaps, it will go around the obstacle forever, because no gap chasing movement modifies the information state of the robot.

T are marked as nonprimitive. This is because the robot has not yet seen what is behind the corresponding gaps. The robot can arbitrarily choose to chase any one of the gaps, and one of the following will occur when it crosses the critical event related to the gap: (1) the gap will disappear, or (2) the gap will split. If the gap disappears, this means that the robot has seen the entire portion of the environment that was originally behind the gap. In this case, the robot must choose another nonprimitive gap to chase. If the second condition occurred, and the gap splits, then the robot can choose to chase either one of the new gaps. This process can be applied repeatedly until the first condition is met and a gap finally disappears. There is one further complication. In some cases, there may be a nonprimitive leaf node, d, that is not a child of the root. To handle this situation, the robot must first chase the child of the root that is an ancestor of d in T. Once a split occurs, it must then follow the next ancestor. This process repeats until d finally disappears or splits. Note that during motion, critical events may occur in other gaps, particularly appearances and merges, and T must be updated accordingly. It is worth nothing that only the angular order, but not the precise angular position of the gaps, is taken into account to build the tree. The construction of T ends when all of the leaves are marked as primitive. This condition indicates that the robot has seen the whole environment.

3.3 The GNT for multiply-connected environments

We briefly mention the complications that multiply-connected environments present to the gap sensor model. For a more extensive description, the reader is referred to [31]. The most interesting complication is that for some environments there may be no critical events at all. Consider Figure 3.b, where the environment's boundary does not have either inflections or bitangents. If the robot tries to follow either one of the gaps, expecting it to disappear,

it will go around the obstacle forever. In terms of information states, there is no gap chasing movement that modifies the current information state of the robot. Even if there were critical events, no gap topological change gives the information to the robot that it surrounded the obstacle once already. To solve this, an *artificial* critical event must be introduced, that indicates to the robot that the obstacle has been surrounded already. This serves as a *pebble*, or marker, that the robot drops and picks as needed.

Another complication is present. In general, the obstacles cannot be discriminated one from the other with only gap sensing information. If the obstacles are assumed to be uniquely identifiable (i.e., they have different *color*), a second sensor can indicate to the robot which gaps have been already explored. For example, a gap may be associated with the obstacle where it begins (the 'near' of the transition). A simple strategy to construct T may command the robot to 'wall follow' each of the obstacles of the environment. In this case, the environment is completely explored when each of the obstacles has been surrounded. This strategy assumes that the robot has the capability of navigating by wall following, in addition to navigating by chasing gaps.

4 Using the GNT: Solving some visibility-based tasks

4.1 Robot Localization

For brevity, we only sketch a straightforward algorithm for robot localization. Given a map of the environment, the cells formed by the inflection rays and bitangent complements can be computed. For simplicity, let the state space X be the set of all of these cells. For each cell, a shortest path tree can be obtained using a breath-first search in the bitangent graph. To each of the nodes of the shortest-path tree a label of "L" or "R" is assigned. The label will indicate the side to which the environment's boundary lies, with respect to an observer standing at the node, and looking away from the root. A localization strategy here will try to match a partial construction of the T with a shortest-path tree of one of the cells. Note that the GNT can be obtained from a shortest-path tree by following the methods to find visibilty polygons from shortest-path trees presented in [1].

At the beginning, the possible position of the robot belongs to $X_0 = X$. After the first sensor reading, a partially constructed T, T_0, is used to compute $X_1 = F(X_0, T_0)$. The set X_1 consist of all the cells to which the sequence of 'L' and 'R' of the gap sensor reading correspond with the first level of nodes of the respective shortest-path trees. As the robot moves, a set of possible positions X_i is computed from the partial constructed T_i, until a set X_n, where no robot movement can decrease the position uncertainty. If $|X_n| = 1$, the robot will be localized, up to a visibility cell, and up to a rotation of the environment.

To reduce the position uncertainty, the localization movements can be computed at least in two ways. The first one is using a modified version of the spiral search in the Kleinberg's algorithm [17], using the gaps as the incident edges of a tree representation of the environment. The movements will be similar if the strategy presented by Dudek et. al.[7] is used. Instead of using the arrangement of visibility cells, the shortest-path trees are used for the comparisons in the movement decisions. Note that the localization strategy sketched here works with only the information provided by the edges not used in the visibility skeleton, which is the opposite case from previous approaches.

4.2 Optimal and locally optimal navigation

We state the optimality of the paths encoded in T by chasing sequences of gaps with the following two propositions:

Proposition 1 *Let $q \in F$, and assume that F is simply-connected. If $V = (v_1, v_2, ..., v_n)$ is the sequence of tangency points of ∂F of the shortest path between q and v_n in the bitangent graph, and $G = (\alpha_1, \alpha_2, ..., \alpha_n)$ is the sequence of gaps where α_i is the gap produced by the tangency point v_i when it is visible, then the path generated by chasing the gap sequence G is the shortest path between q and v_n.*

Proof. It is enough to prove that the path between v_i and v_{i+1} is optimal, since the sequence V is optimal by definition. The shortest path between two points in the Euclidean plane is unique and is a straight line. Following α_{i+1} the trajectory is a straight line that ends in v_{i+1}. We conclude that the path between v_i and v_{i+1} is optimal. □

Proposition 2 *The paths encoded in T between the root of T and any point $q \in F$, if F is simply-connected, are optimal.*

Proof. Let $p \in F$. Let $T(p)$ refer to the configuration of T when the robot is at p. From $T(p)$ we generate the sequence of gaps $G = \{\alpha_1, \alpha_2, ..., \alpha_n, \alpha_q\}$ that must be chased to reach q. By following G, the tangency points $V = \{v_1, v_2, ..., v_n, q\}$ of ∂F are visited. Let G_o be the sequence of gaps that generates the shortest path between p and q. Let $\alpha_d \in G$ (of point $v_d \in V$) be the first gap in which G and G_o differ. Since critical events are recorded in T as they become visible, this means that the point v_d is visible before the rest of the path in G_o, which therefore, *hides* behind α_d. The shortest path between the current position of the robot and the rest of the path encoded in G_o is the one following α_d (by Proposition 1). Therefore, G_o contains α_d, and we conclude that $G = G_o$. □

By Proposition 2, the paths encoded in T are optimal, and in this sense, a shortest-path tree and the GNT encode the same path information. Optimal

navigation occurs by executing motions that follow the path in T from the root to the desired node d. During these motions, the robot records all changes to T that occur from critical gap events. At every time during the navigation, the robot chases the gap that corresponds to the child of the root that is an ancestor of d. At several points, the gap may split, and the robot must chase the new child of the root that is an ancestor of d. This results in a sequence of gap chasing commands, which ends when d is not detected anymore. As it is presented in [32], *interesting* objects in the environment can be associated with gaps, and the robot can manipulate (i.e., move) them through optimal paths.

When T is constructed for a multiply-connected environment, global optimality cannot be guaranteed. This is because T is a tree, which encodes only one path between the robot's position and any place in the environment. Since no distance information was taken into account in T, the path encoded may not be the optimal one. When multiple paths are found to the same place, a heuristic is used and only the paths with less gaps to chase are kept. This guarantees that the robot will travel through less cluttered areas, but tells nothing about optimality. Still, by a similar argument used in Proposition 1 they are still locally optimal. For more details, the reader is referred to [31].

4.3 Pursuit-Evasion

We next consider augmenting the GNTs to handle pursuit-evasion. A gap may be labeled as *cleared*, *contaminated* or *recontaminated* as a function of the region that hides behind it. Initially, all of the gaps in T are labeled as contaminated. If a gap appears, it is labeled as cleared, since if an evader is behind the gap, the pursuer would already have detected it. A cleared gap is recontaminated if it merges with a contaminated or a recontaminated gap. If T has at least one node that is not cleared, it is labeled as contaminated; otherwise, it is said that T is cleared. Since each node in T encodes a connected region of the environment, solving the pursuit-evasion problem is equivalent to clearing each node of T.

In [14] we present an online strategy that reads "cleaning" schedules from T. Each schedule is an ordered sequence of gaps that should be cleared. In other words, it is a sequence of changes in the visibility region that guarantees that some region of the environment is cleared. The strategy works by growing a set of adjacent children of the root of T. When a gap recontamination occurs, this is naturally reflected in T by the merge critical event. Schedules are then defined recursively, where for a node encoding a merge, its children should be cleared first. Circular dependencies of contamination merges can be detected, as in [25], to conclude that the environment cannot be successfully cleared with a single pursuer. The pursuer can place a static *sentry*, or guard, to prevent a critical event from happening, breaking the

circular dependency. As proved in [14], by using a GNT the number of sentries needed is asymptotically optimal, $O(\log n)$, where n is the number of bitangents in the environment.

Although the GNT keeps all the contamination information to solve a pursuit-evasion problem, it may not encode the paths needed to clear some regions. The GNT only provides shortest-paths, but maybe longer paths are needed to avoid some critical event (i.e., a merge) from happening. Figure 4 shows an environment that can be cleared by a single pursuer, but not by following paths in the GNT.

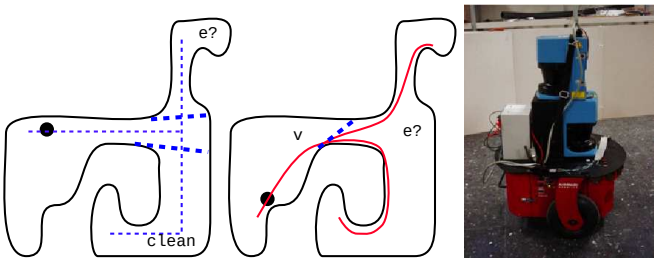


Fig. 4. (a) Not every environment searchable with one robot is searchable with one robot following sequences of gaps. A robot following the dashed path can find all of the evaders (left) . The thick paths, generated by chasing gaps, cause unavoidable recontaminations (right). (b) Robot platform used in the real experiments. The gap sensor was realized using two laser range sensors.

5 Implementation and conclusions

The GNT construction for a simply-connected environment was successfully implemented using the robot platform shown in Figure 4.b.. In the GNT model, the robot was considered as a point that can get arbitrarily close to the boundary before a critical event is triggered. Since this is not possible in the real robot, wall-following capabilities were provided such that the robot still chase the detected gaps, while taking into account the environment's boundary.

Although from a planning perspective the GNT may unify some of the visibility tasks, the navigation paths it offers may not be adequate for all robotic settings. For example, one may prefer the safety of maximum clearance paths [5], instead of the shortest-paths. Still, the GNT may be used to direct the motions of a robust navigation system, such as the one presented in [21]. An algorithm using the GNT can determine which discontinuity to

follow, and the navigation system, at a lower level, will determine the safer motions for the robot.

Acknowledgements: The authors want to thank Boris Simov, Stjepan Rajko, Shai Sachs and Rafael Murrieta for their helpful discussions on the gap sensing model, to Javier Minguez for his comments on navigation on real robots, and to Anna Yershova and Robert Ghrist for their interesting discussions on information spaces. The robot equipment was provided by the ITESM Campus Ciudad de México and by the ITESM Campus Morelos, México. This work is supported in part by ONR Grant N000014-02-1-0488 and NSF-CONACyT Grant 0296126.

References

1. B. Aronov, L. Guibas, M. Teichmann, and L. Zhang. Visibility queries in simple polygons and applications. *Algorithms and Computation, 9th International Symposium, ISAAC '98*, 1533, 1998.
2. 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.
3. J. Borenstein, B. Everett, and L. Feng. Navigating mobile robots: Systems and techniques. A.K. Peters, Wellesley, MA, 1996.
4. K. W. Bowyer and C. R. Dyer. Aspect graphs: An introduction and survey of recent results. *Int. J. of Imaging Systems and Technology*, 2:315–328, 1990.
5. H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Int. Conf. Robot. & Autom.*, 17(2):125–137, April 2001.
6. B. R. Donald. On information invariants in robotics. *Artif. Intell.*, 72:217–304, 1995.
7. G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 437–446, 1995.
8. F. Durand. *3D Visibility: Analytical study and applications*. PhD thesis, Université Grenoble I – Joseph Fourier Sciences et Géographe, July 1999.
9. M. Erdmann. Understanding action and sensing by designing action-based sensors. *Int. J. Robot. Res.*, 14(5):483–509, 1995.
10. M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Trans. Robot. & Autom.*, 4(4):369–379, August 1988.
11. K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.
12. 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.
13. L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. 1st Workshop on Algorithmic Foundations of Robotics*, pages 269–282. 1995.

14. L. Guilamo, B. Tovar, and S. M. LaValle. Pursuit-evasion in an unknown environment using gap navigation graphs. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2004. Under review.
15. 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.
16. I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. *IEEE Trans. Robot. & Autom.*, 13(6):814–822, December 1997.
17. J. M. Kleinberg. The localization problem for mobile robots. In *IEEE Symposium on Foundations of Computer Science*, pages 521–531, 1994.
18. J.-C. Latombe. *Robot Motion Planning*. Boston, MA, Kluwer Academic Publishers, 1991.
19. S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–201, April 2001.
20. 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.
21. J. Minguez and L. Montano. Nearness diagram navigation (ND): Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, February 2004.
22. N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *1st International Joint Conference on Artificial Intelligence*, pages 509–520, 1969.
23. J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, 1987.
24. C. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.
25. S.-M. Park, J.-H. Lee, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searcher. Technical Report CS/TR-2001-161, KAIST, Dept. of Computer Science, Korea, January 2001.
26. M. Pocchiola and G. Vegter. The visibility complex. *Int. J. Comput. Geom. & Appl.*, 6(3):279–308, 1996.
27. S. Rajko and S. M. LaValle. A pursuit-evasion bug algorithm. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1954–1960, 2001.
28. E. Rimon. Construction of C-Space roadmaps from local sensory data. What should the sensors look for? *Algorithmica*, 17(4):357–379, 1997.
29. S. Sachs, S. Rajko, and S. M. LaValle. Visibility-based pursuit-evasion in an unknown planar environment. Submitted to *International Journal of Robotics Research*, 2003.
30. I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. Computing*, 21(5):863–888, October 1992.
31. B. Tovar, S. M. LaValle, and R. Murrieta. Locally-optimal navigation in multiply-connected environments without geometric maps. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 3491–3497, 2003.
32. B. Tovar, S. M. LaValle, and R. Murrieta. Optimal navigation and object finding without geometric maps or localization. In *Proc. IEEE International Conference on Robotics and Automation*, pages 14–19, 2003.