Distance-Optimal Navigation in an Unknown Environment without Sensing Distances

Benjamín Tovar, Rafael Murrieta-Cid, and Steven M. LaValle

***Abstract*— This paper considers what can be accomplished using a mobile robot that has limited sensing. For navigation and mapping, the robot has only one sensor, which tracks the directions of depth discontinuities. There are no coordinates, and the robot is given a motion primitive that allows it to move toward discontinuities. The robot is incapable of performing localization or measuring any distances or angles. Nevertheless, when dropped into an unknown planar environment, the robot builds a data structure, called the Gap Navigation Tree, which enables it to navigate optimally in terms of Euclidean distance traveled. In a sense, the robot is able to learn the critical information contained in the classical shortest-path roadmap, although surprisingly it is unable to extract metric information. We prove these results for the case of a point robot placed into a simply connected, piecewise-analytic planar environment. The case of multiply connected environments is also addressed, in which it is shown that further sensing assumptions are needed. Due to the limited sensor given to the robot, globally optimal navigation is impossible; however, our approach achieves locally optimal (within a homotopy class) navigation, which is the best that is theoretically possible under this robot model.**

***Index Terms*— Visibility, navigation, optimality, map building, minimal sensing, shortest paths, information spaces, sensor-based planning, bug algorithms.**

I. Introduction

In the design of many mobile robot systems, the intuition is often that "more information is better". This typically leads to the integration of powerful sensors that provide dense, accurate measurements of distance information. The goal is typically to construct a complete geometric map of the robot's environment while localizing the robot with respect to its map [40]. As the number of sensors and the amount of data increase, there are substantial burdens in terms of cost, power consumption, reliability, and modeling. Therefore, our work investigates the *minimal* information that is needed to solve some tasks. By establishing that certain tasks can be solved using simple sensors, it may be possible to avoid costly sensors and substantial modeling challenges. Perhaps "less information is better".

We model the robot as a point moving in an unknown planar environment. The robot is assumed to have an abstract sensor (in the sense of [14]) that reports the order of depth discontinuities of the boundary, from the current position of the robot. These discontinuities are called *gaps*, and the abstract sensor may be implemented in a number of ways, including using an omnidirectional camera or a low-cost laser scanner. To characterize its environment, the robot builds a dynamic data structure, called the *Gap Navigation Tree* (GNT), entirely from online sensor measurements. 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 to maintain shortest-path information from the current position of the robot. These paths are globally optimal in Euclidean distance traveled if the environment is simply connected, even though geometric information, such as lengths, angle measurements and robot orientation, is not available.

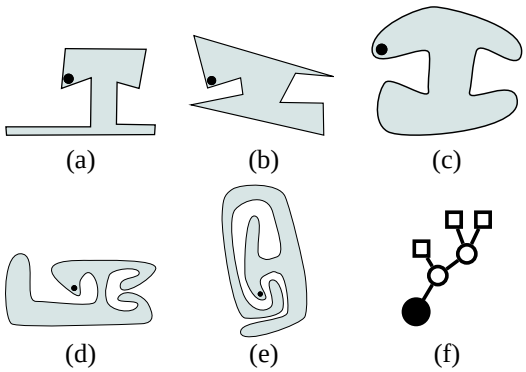


Fig. 1. Figures (a) to (e) show environments which are indistinguishable to the robot (black disc) with limited sensing; however, it can still navigate optimally using the Gap Navigation Tree shown in (f).

Our approach is based on the careful consideration of information spaces. To illustrate, consider Figure 1. Using the sensing and action history available to the robot, the five environments are indistinguishable, and generate the same GNT, shown in Figure 1.(f). Their scale and orientation are also unknown. All such environments fall into an enormous equivalence class (called a nondeterministic I-state in [26]). Surprisingly, the robot can perform optimal navigation without trying to resolve these ambiguities. Thus, the sensing model is close to the information requirements of the task.

The current paper is derived from previous conference publications in [41]–[43].

II. Previous work

In our work, the data structure generated by the robot can be considered as a topological map, as opposed to grid-based

Manuscript submitted as a regular paper

B. Tovar and S. M. LaValle are with the Dept. of Computer Science, University of Illinois, Urbana-Champaign.

R. Murrieta is with the Center for Mathematical Research (CIMAT), A.P. 402, Guanajuato, Mexico, 36000.

The corresponding author is B. Tovar. 201 N. Goodwin 3340, Urbana, IL, 61801. email: btovar@uiuc.edu. phone: +1 (217) 244-5972, fax: +1 (217)265-6591

This work was founded by ONR Grant N000014-02-1-0488 and NSF-CONACyT Grant 0296126.

maps [3], [13], [29] or exact geometric representations [17]. A topological map usually takes the form of a graph, in which the vertices represent particular sensor readings and configurations and the edges represent the controls between two different configurations [37]. Our work differs from previous approaches in that we are interested in a *local* representation, defined for the current position of the robot, rather than a global one, like in [5], [6], [8], [10], [39].

Our research considers minimal sensing for mobile robots. This was also considered in *bug algorithms* [20], [21], [28], in which a robot that combines global knowledge with local information is able to navigate among boundary components and reach a known goal. The robot navigation capabilities are simple (movement towards boundary components and wall-following), no representation of the environment is maintained, and the global information consists only of the position of the goal. These characteristics allow the use of bug algorithms in robots that have very limited sensing capabilities and unreliable motion control. More importantly, the memory required for the algorithms is constant.

In general, minimal sensing does not allow the full knowledge of the state. In robotics, the problem of driving a system from an unknown state to a goal state was considered in the context of manipulation [15]. For example, up to convex hull symmetry it is possible to manipulate polygonal parts to a final configuration without any sensor information [16]. Of course, not all robotics tasks can be solved without sensors, but it is very interesting, and scientifically important, to determine the minimum information necessary to complete a given task [2], [9]. Moreover, one may go a step further and design a sensor that exactly suits the robotics task. One can think of an *abstract sensor* that gives the "ideal" minimal information to the robotics system to work correctly, and its physical implementation using a "subideal" sensor [14].

One such abstract sensor reports the set of points visible from the current position of the robot. This sensor gives the *visibility region*, which formally is defined as the set $V(x)$, in which x is the position of the robot, and $q \in V(x)$ implies that the open line segment joining x and q does not intersect the environment boundary. As a robot moves, the visibility region changes, modifying the information about the environment or its progress towards a goal. The changes in the visibility region have been extensively studied, from the art-gallery problem [32], to decompositions of the environment into regions of *similar* visibility. In [34], 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 environment can also be decomposed also into equivalence classes of similar visibility of an object. Elements inside a class have a similar qualitative view of the object: they see the same *aspect*. An aspect is defined as the set of views of an object that share the same combinatorial structure. This leads to the *aspect graph* [4], [24]. In [18], a planar environment is decomposed into cells that *see* the same aspect of the environment boundary. Such a decomposition is called the *visibility cell decomposition*, and each cell is called a *visibility cell*.

In the decompositions mentioned before, as the robot moves inside of a cell, there is no significant change in information. The robot receives the same combinatorial information from the sensors. In contrast, as the robot crosses a cell boundary, the combinatorial structure of the visibility region drastically changes, and the robot's information may be modified. Such sudden changes are called *visual events* [12]. Our paper focuses on the use of visual events for optimal navigation in the plane.

Finally, a similar data structure to the GNT was presented in [1], where the shortest-path tree is updated when a point crosses *constraint lines*. For this approach, complete knowledge of the polygon where the point is moving is assumed, which corresponds to exact localization and perfect sensor measurements. The focus of that work was to compute online changes in the visibility polygon of an observer in motion.

III. Robot Model

The robot is modeled as a point moving in an unknown *environment*, which could be any compact set $R \subset \mathbb{R}^2$ for which the interior of R is connected, and let the boundary of R, ∂R, be the image of a piecewise-analytic closed curve. Note that R is simply connected, which is an assumption that will be removed in Section VI.

A. The gap sensor

The robot has only one sensor, called a *gap sensor*, which is only able to detect and track discontinuities in depth information. The gap sensor is an *abstract sensor* [14], which means that its physical implementation may vary. It can be imagined as a crude range sensor that gives inaccurate distance information, but from which a kind of edge detector can be used to extract the discontinuities in the measurement. This sensor can be implemented with a laser range finder, sonars, cameras, or with an ad hoc sensing system. For example, imagine an inexpensive laser pointer rotating rapidly horizontally on the robot position, so that a horizontal line is drawn in the field of view of the robot. An omnidirectional camera can detect where this line *breaks*, thus detecting the discontinuities in depth information. The robot does not have any geometric information about the discontinuities, other than their cyclic ordering with respect to the robot's local frame of reference.

Each discontinuity will be referred to as a *gap* [35], [38], which also corresponds to a region of R that is not visible to the robot[1]. For example, Figure 2 (right) shows the gaps for the environment shown in Figure 2 (left). It is assumed that the robot can track and distinguish the gaps at all times and record any of their combinatorial changes.

Let $G(x) = [g_1, \ldots, g_k]$ denote the sequence of gaps as they appear in the gap sensor when the robot is at $x \in R$. If x lies in the interior of R, then $G(x)$ is a cyclic ordering; therefore, due to being cyclic, statements such as $[g_1, \ldots, g_k] = [g_2, \ldots, g_k, g_1]$ can be made. If $x \in \partial R$, then part of the sensor view is obstructed by the boundary, and a linear ordering of gaps is obtained; however, this extra

[1]The gaps correspond to the spurious edges defined in [18].

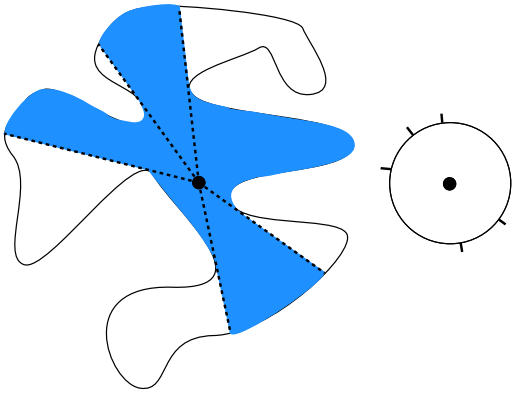
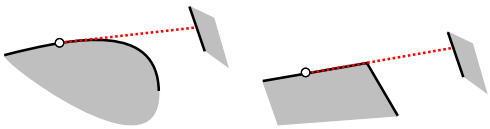


Fig. 2. The robot's view of the environment. The position of the robot is shown with a black disk. On the left, the environment and the visibility region of the robot. On the right, angular position of the gaps detected in the visibility region.



(a) (b)

Fig. 3. Gaps from the environment's boundary. On (a), the boundary to the right is smooth and curves below from the right tangent ray. On (b), the degenerate polygonal case is shown.

information will not be important and is not necessarily known to the robot. It is important to define which gaps appear if $x \in \partial R$. A gap will appear to the right (counterclockwise direction along ∂R) of x if either of the two cases shown in Figure 3 occurs. In Figure 3.(a), the boundary to the right is smooth and curves below from the right tangent ray. Figure 3.(b) shows a degenerate polygonal case in which the boundary remains on the right tangent ray and then curves below (either smoothly, or abruptly at a nonsmooth point). The potential gap to the left of x is defined the same way. Note that if the robot moves across a nonsmooth point on the boundary, the boundary gaps can jump discontinuously. We assume these are nevertheless tracked because all of the other gaps moved continuously, and the proper correspondence can be made.

For any $x \in R$, each $g_i \in G(x)$ is merely a unique label, and does not contain information about lengths or angles. Most often, as the robot moves a small amount, the gap sequence does not change. Occasionally, fundamental changes occur, such as gaps appearing, disappearing, merging, or splitting; these cases will be covered in detail shortly. Suppose that the robot moves along any path, $\tau : [0,1] \rightarrow R$. If there is some gap g for which $g \in G(\tau(s))$ for all $s \in [0,1]$, then it is assumed that the robot maintains its unique label. Thus, the robot is not confused about the identity of any gaps that remain in the gap sequence as it moves. However, if some gap disappears and then reappears later, we do not require the robot to recall of the old label. Thus, there is no registration problem.

Recall that the robot has no previous knowledge of R and it is not capable of building an exact map of the environment.

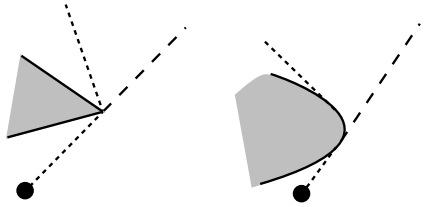


Fig. 4. The chase$(\cdot)$ motion primitive. When the boundary is not smooth, the robot may not necessarily transverse an interval of ∂R when executing a chase$(\cdot)$ motion primitive.

The robot has no compass, odometers, or other sensors that might be used to derive distances.

B. Motion primitives

Since the robot does not have access to coordinates, it is important to define a control model that does not require them. The robot motions are expressed as a sequence of *motion primitives*, which are described solely in terms of information from the sensor. This enables motions to be expressed without referring to coordinates in $\mathbb{R}^2$. For a gap $g \in G(x)$, a *gap chasing* motion primitive is denoted as chase(g); note that there is no reference to x because this is unknown to the robot. In this motion primitive, the robot rotates to align its heading with the gap and moves forward with unit speed. The robot uses sensor feedback to continue the motion, which is guaranteed to be collision free, except for tangential motions along the boundary. If the robot trajectory would be directed into the interior of the complement of R, then the gap would not have appeared in $G(x)$. Note that chase(g) might cause the robot to follow the boundary, as shown in Figure 4.

The motion primitive can be considered as an action in a hierarchical approach. It is therefore important to specify the conditions under which a motion primitive terminates. Let $\tau : [0, t_f] \rightarrow R$ denote the trajectory taken by the robot when executing a motion primitive chase(g). Consider how $G(\tau(t))$ evolves as the robot moves. It should be possible to chase g only when it is present in $G(\tau(t))$. Therefore, chase(g) terminates when g disappears from $G(\tau(t))$. This termination is guaranteed by Lemma 2, which is presented in Section IV. All robot motions are based on primitives. Therefore, any motion strategy for the robot must be a finite sequence of primitives.

IV. The Gap Navigation Tree

Suppose that the robot moves along any path, $\tau : [0,1] \rightarrow R$. Consider the information obtained from the gap sensor. For every $s \in [0,1]$, a cyclic sequence, $G(\tau(s))$, of gaps is observed. In this section, we will define a compact representation of information that is relevant for optimal navigation and appears in $G(\tau(s))$ for all $s \in [0,1]$.

A. Compressing the sensor history

Suppose that initially, the gaps are labeled consecutively, starting with g_1. For each new gap that appears, it is assigned

the next unused integer in $\mathbb{N}$ to ensure uniqueness. To consider the problem of maintaining information, suppose that the robot moves along some path τ, and initially, $G(\tau(0)) = [g_1, g_2, g_3]$. Now suppose that at some $s' \in (0,1)$, gap g_2 splits into two new gaps, which are labeled by convention as g_4 and g_5. The gap sensor reads $G(\tau(s')) = [g_1, g_4, g_5, g_3]$. Now suppose that at some $s'' \in (s', 1)$, gaps g_4 and g_5 merge into a single gap. By convention, g_6 is assigned, rather than worrying about correspondence to the original gap labeled g_2. It will be useful for navigation purposes to remember that g_4 and g_5 were merged to obtain gap g_6. Perhaps this could be encoded as $G(\tau(s')) = [g_1, [g_4, g_5], g_3]$, in which $[g_4, g_5]$ is used as the gap label instead of g_6. This idea could be applied iteratively to make complicated, nested expressions for the structure of merged gaps that appear in $G(\tau(1))$. Note that the cyclic order of $G(\tau)$ is enough to generate the correct label correspondences when a gap splits. In the previous example, this means that when gap g_6 splits, of the new two gaps that are detected, the one detected after gap g_1 is actually gap g_4. Also note that given a geometric constraint (Lemma 1 in Section IV-C), the correspondence between gaps g_2 and g_6 can be determined, but such information is not currently needed.

Rather than representing merge information syntactically, it will be convenient to express it as a rooted tree in which all children are ordered. Suppose once again that a path $\tau : [0,1] \rightarrow R$ is executed. Let the *Gap Navigation Tree (GNT)* be a rooted tree, defined as follows. Every non-root vertex of the GNT is a gap that appears in $G(\tau(s))$ for some $s \in [0,1]$. Every child vertex of the root is a gap in $G(\tau(1))$, and they are cyclically ordered around the root in the same way that they appear in $G(\tau(1))$. All remaining vertices (i.e., not the root and its children) in the GNT are gaps that appeared in $G(\tau(s))$ for some $s < 1$, but not appear in $G(\tau(1))$ due to merging. The children of any non-root vertex, v, are precisely the gaps that were merged to form v, and are assumed to be ordered in the same way that they once appeared in the gap sensor.

When is a GNT as complete as possible for a particular environment? This question will be addressed in detail shortly; however, it is convenient to have the definition now. Consider the leaf vertices of a GNT. If any leaf vertex has the potential to split, then the GNT is incomplete because it could expand. Recall that some gaps split when approached using chase(g) and others simply disappear. Let the gaps that disappear and their corresponding vertices in the GNT be called *primitive*. If all leaves of a GNT are primitive, then the GNT is said to be *complete*. A geometric interpretation of this will be given in Section IV-C.

The primary use of the GNT is to define a sequence of motion commands that guides the robot to a gap that once was once observed by the gap sensor and is consequently not a child of the root. Let g be such a gap. It appears in the GNT because it was involved in one or more gap merges. The merges can be undone by applying the chase$(\cdot)$ primitive to every gap in the GNT that is an ancestor of g. Suppose, for example, that the path from the root to g is $(g_1, g_2, \ldots, g)$ (ignoring the root, which is not a gap). The primitive chase(g_1) forces g_1 to split, which enables chase(g_2) to be applied. This process is applied inductively until g is observed by the gap sensor, and chase(g) can be applied. Let $\mathbf{chase}(g)$ denote the corresponding sequence of motion primitives. We may therefore say that any gap in the GNT can be *chased*, which means that a sequence of motion primitives is executed until the gap is eventually chased.

B. Critical events and incremental GNT construction

The GNT can be constructed incrementally as the robot moves along a path τ. Initially, the GNT consists of a root vertex that is connected to one leaf vertex for every gap in $G(\tau(0))$. Each time t at which a change in $G(\tau(t))$ occurs corresponds to a *critical event*. This requires updating the GNT. There are four different kinds of critical events (see Figure 5):

1) *A new gap g appears:* A vertex g is added as a child of the root, while preserving the cyclic ordering from the gap sensor.
2) *Gaps g_1 and g_2 merge into g:* Vertices g_1 and g_2 become children of a new vertex, g, which is added as a child of the root and preserving the cyclic ordering.
3) *A gap g disappears:* The vertex g, which must be a leaf, is removed.
4) *Gap g splits into g_1 and g_2:* If g is a leaf vertex, then g_1 and g_2 become new vertices; otherwise, they already exist as children of g. Both g_1 and g_2 are connected to the root, preserving the cyclic ordering and removing g.

C. Geometric interpretation of the GNT

Now consider the geometric information that can be inferred about the environment from the GNT. This will help us to prove that the GNT enables optimal navigation. We begin the discussion with the relation between critical events and the geometry of the environment. Critical events are determined by *generalized inflections* and *generalized bitangents* of ∂R (see Figure 6). Following the presentation of [27], a generalized inflection of ∂R is identified with a connected, open set $I \subset \partial R$ if there exist a line L that partitions I into three connected sets I_1, I_2 and I_3 such that: 1) I_1 is an open set that does not intersect L, 2) I_2 is a closed subset of L, and 3) I_3 is an open set that does not intersect L, and that lies on the opposite side of L from I_1. If I_2 is a single point, then the right derivative of ∂R (taken in the limit of open intervals in I_3) evaluated at I_2 corresponds to the slope of L. Likewise, a pair of disjoint connected open sets I and J identify a generalized bitangent if at least one point of I is visible to one point of J, and if there is a line L that partitions I and J into sets I_1, I_2, and I_3, and J_1, J_2, and J_3 respectively, such that: 1) I_1 (J_1) is an open set that does not intersect L, 2) I_2 (J_2) is a closed subset of L, and 3) I_3 (J_3) is an open set that does not intersect L, and that lies on the same side of L from I_1 (J_1). From now on, when we write *inflection* or *bitangent*, we mean *generalized inflection* and *generalized bitangent*, respectively.

Given an inflection, identified by line L and sets I_1, I_2 and I_3, as defined before, an *inflection ray* is found by extending a ray inside R with the same slope as L, from a point in I_2 until a point of ∂R is hit. Given a bitangent, identified by

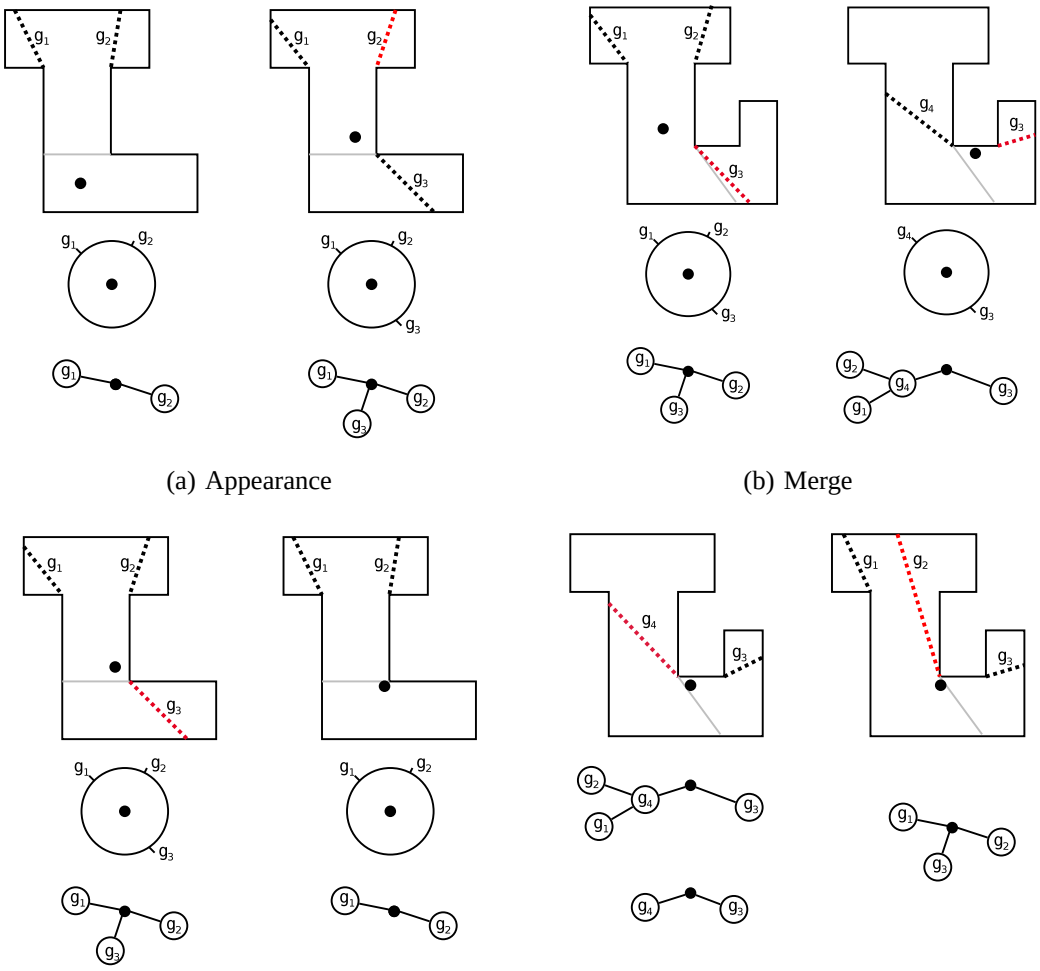


Fig. 5. Updates in the Gap Navigation Tree. The relevant inflection or bitangent complement is indicated with a gray line segment. (a) Chasing gap g_1, gap g_3 appears, and the respective vertex is added to the root. (b) When gap g_3 is chased, gaps g_1 and g_2 merge. They become children of a new gap g_4, and the tree is updated accordingly. (c) Gap g_3 disappears when chased. The vertex corresponding to g_3 is removed from the tree. (d) Gap g_4 splits into gaps g_1 and g_2. The two cases for the split are presented here: If g_4 was known to have descendants, these become children of the root; otherwise two new vertices are created.

the sets I_1, I_2, I_3, J_1, J_2, J_3, and by the line L, a *bitangent line segment* is any of the open segments with endpoints in I_2 and J_2 completely contained in R. For each bitangent, two *bitangent complements* are defined. These correspond to the two rays with the same slope of L, starting at a point in I_2 and J_2, extending until ∂R is hit, and not containing any point of the bitangent line segments of the corresponding bitangent.

Inflection rays and bitangent complements decompose R into cells of *similar visibility*, called aspect cells (also called visibility cells). We use a common general position assumption that no line is tangent to more than two points of the boundary, since such *tritangents* would not survive a small deformation of the environment [23]. Without this general position assumption, the simple capabilities given to the gap detector make it impossible to distinguish some critical events that occur simultaneously. For example, a gap splitting into three gaps from a gap splitting into two gaps occurring very close and simultaneously to a gap appearance. As illustrated in Figure 6.(a), appearances and disappearances of gaps occur when the robot crosses inflection rays. The other critical events, merges and splits of gaps, are related to bitangent line segments of ∂R. Merges and splits occur when the robot crosses bitangent complements (Figure 6.(b)).

Together with the previous discussion, the following lemma is presented:

Lemma 1: Let g_1 and g_2 be two gaps that merge into gap g_3. When g_3 splits, g_1 and g_2 appear at the same angular position in R at the time of the merge, independently from the robot's motion.

Proof: Merges and splits occur when the robot crosses a bitangent complement of ∂R. Thus g_1, g_2, and g_3 are aligned with the bitangent at the split or the merge. This is independent of where the bitangent complement is crossed. □

The previous lemma associates two critical events to a particular bitangent complement: a split and a merge. This is the basis for the correct encoding of critical events in the GNT. Even though the robot may not be able to recognize that a gap that appears was detected previously, Lemma 1 implies that

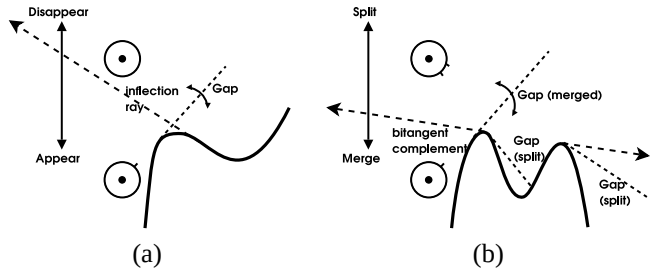


Fig. 6. Critical events. (a) Appearance and disappearance of gaps occur when the robot crosses inflection rays. (b) Splits and merge occur by crossing bitangent complements.

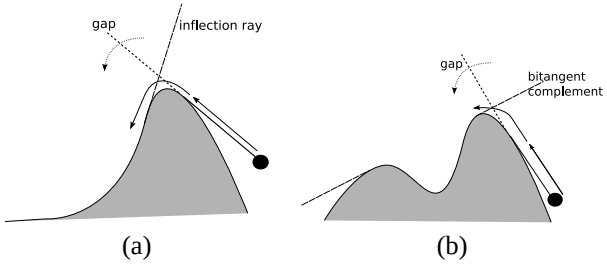


Fig. 7. Every chase$(\cdot)$ motion primitive terminates, either with a disappearance or with a split critical event. The black disc represents the position of the robot when the chase$(\cdot)$ motion primitive is issued. After reaching ∂R, the robot moves tangentially to ∂R, until an inflection ray (a) or a bitangent complement (b) is crossed.

when a gap splits, it can only split into gaps that merged before. Lemma 1 also provides the correct label correspondences after a gap split. The identification of the gaps is done purely by the order of the gaps before the corresponding merge, and not by the "features" of the environment that produced them.

D. The gap-based roadmap

Consider a path segment $\tau : [0, 1] \rightarrow R$ followed by the robot with the motion primitive chase(g). Assume now that the chase(g) has not been issued, but that the robot is at a point $\tau(t)$, with $0 < t < 1$. If chase(g) is issued, then the robot follows the same path τ, restricted to $[t, 1]$. This is because gap detection depends only on the current position of the robot. Consider now the paths followed by each of the possible motion primitives after a critical event. The set of points of R visited by all such paths is called the *gap-based roadmap*, and it is denoted by S. Once the robot is on a point of S, chase$(\cdot)$ motion primitives may only reach points inside S. All of the paths that generate the roadmap have finite length, which is is a direct consequence from the following lemma:

Lemma 2: Termination of chase(g) is guaranteed for any $g \in G(x)$ and any $x \in R$, and is caused by only two possible critical events: disappearance or splitting of g.

Proof: The heading of the robot is always aligned with g, which forbids the robot to follow any cycle, or to move away from g. Consider now Figure 7. As the robot moves with unit speed towards the gap, the starting point of the gap slides on ∂R, towards the respective inflection ray or bitangent complement. When the robot reaches ∂R, the position of the robot and the starting point of the gap coincide, and three cases should be considered: 1) the robot moves away from the inflection ray or bitangent complement, 2) the robot is stationary in ∂R, 3) the robot moves towards the inflection ray or bitangent complement. Cases 1 and 2 cannot occur, since the heading of the robot always points to the gap, and the robot moves tangentially on ∂R with unit speed. Thus, the remaining case always occurs, and the respective inflection ray or bitangent complement is eventually crossed. It is clear that the termination critical event cannot be an appearance, since the gap is already detected ($g \in G(x)$). The critical event cannot be a merge either, because the corresponding merging gap for the bitangent complement pair is not yet visible. □

E. Constructing a complete GNT

Now that we have specified how the GNT is expanded as the robot follows a fixed path, the next task is to determine what motion commands should be executed so that the robot follows a path that builds a complete GNT. Incompleteness of the GNT is caused by any nonprimitive leaves. Therefore, the GNT is forced to be complete by iteratively chasing leaves. Each time that a leaf splits, one of its children can be arbitrarily chosen and chased. If a leaf disappears, then another nonprimitive leaf is selected for chasing. The order in which the nonprimitive leaves is chased is not important. Eventually, all leaves will be primitive, in which case the GNT is complete.

As an example of constructing a complete GNT, suppose the robot is in the environment as shown in Figure 8. In Figure 8.(a) we show the boundaries of the aspect cells. The root of the GNT is shown as a solid black disk. Vertices that are not known to be primitive are shown as circles, and vertices that are primitive are squares. The robot begins to build the GNT as shown in Figure 8.(a). There the robot first executes chase(g_1). When this gap is followed, the robot triggers an appearance event, and gap g_3 is added to the tree (Figure 8.(b)). Later, gaps g_2 and g_3 merge, and they become children of a new vertex, g_4 (Figure 8.(c)). When g_1 disappears (Figure 8.(d)), g_2 is the only remaining nonprimitive gap, and the robot executes $\mathbf{chase}(g_2)$, which generates [chase(g_4), chase(g_2)]. The robot chases g_4 until it splits, and then g_2 is chased (Figure 8.(e)). Finally, when g_2 disappears, all of the leaf vertices are primitives. (Figure 8.(f)).

Lemma 3: The procedure of iteratively chasing nonprimitive leaves terminates with a resulting complete GNT.

Proof: Consider the path τ executed during the procedure. The key observation is that any time that a new gap appears in $G(\tau(s))$, it must be primitive. If the gap is chased, it cannot split. Therefore, the only gaps that contribute to the incompleteness of the GNT are ones that either appeared in $G(\tau(0))$ or were formed by a sequence of splits of these gaps. Even though chasing a leaf may reveal new gaps via splitting, the number of primitive gaps for a given environment is finite because each corresponds to a inflection. There are finitely many inflections because ∂R is piecewise-analytic. Each time that the procedure forces a gap to disappear, it is one step closer to having a complete GNT. Since the number of gaps is finite, the procedure must terminate with a complete GNT.

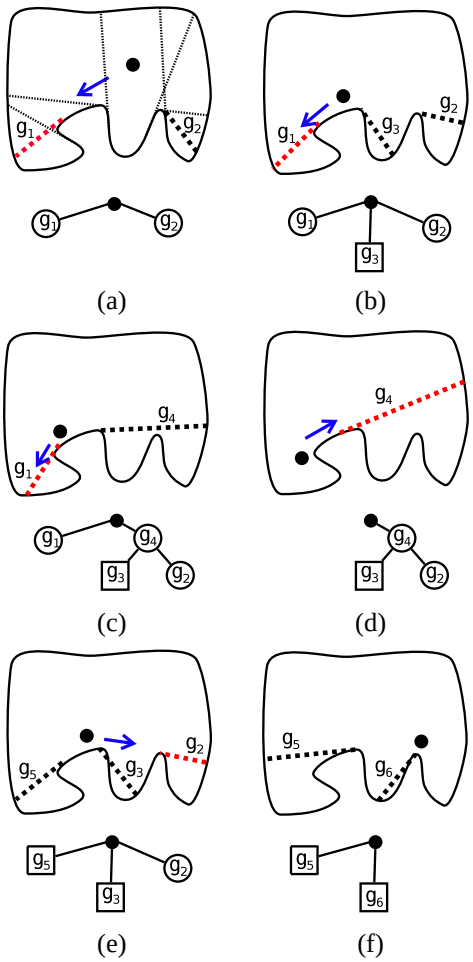


Fig. 8. Building the Gap Navigation Tree. (a) The thin lines show the places where gap critical events are triggered. The robot chases nonprimitive gaps from (a) to (f), updating the GNT accordingly (refer to main text), until all of the leaf vertices are primitive. Squares and circles denote primitive and nonprimitive vertices, respectively.

□

Note that even though a GNT is complete, it nevertheless changes as the robot moves in its environment. This happens because the tree always expresses how the environment appears relative to the local frame of the robot. Once a complete GNT has been constructed, however, it remains complete in spite of any motions executed by the robot.

V. Optimal Navigation

Given that the GNT is built from critical events, and these correspond to the boundaries of the aspect cells in decompositions such as [18], points inside of an aspect cell have the same GNT. In fact, once the GNT is constructed, it encodes the same information as if the single-source, shortest-path problem would be solved from a given aspect cell.

A. Moving along the gap-based roadmap

We now argue the optimality of the paths generated by chasing sequences of gaps in the GNT. In the following discussion we prove the optimality (in the Euclidean sense) using chase$(\cdot)$ motion primitives of paths starting and ending in S. In Section V-C we will extend the optimality arguments to points not in S. For the following discussion, let $p, q \in S$, and let $U = (u_1, u_2, ..., u_n)$, with $u_i \subset \partial R$, be the sequence of maximal connected intervals of ∂R that the robot traverses (in order) in the shortest path from p to q.

Lemma 4: Let $H = (g_1, g_2, ..., g_n)$ be a sequence of gaps, in which g_i is the gap chased when the robot traverses the interval $u_i \in U$. The path generated by chasing iteratively the sequence H is the shortest path between p and q.

Proof: It is sufficient to prove that the path between u_i and u_{i+1} is optimal, since the sequence U is optimal by definition. The shortest path between two points in the Euclidean plane is unique and is a straight line. When the robot transverses the interior of R, from u_i to u_{i+1} when following g_{i+1}, the trajectory is a straight line tangential to u_{i+1}. Finally, the interval u_{i+2} becomes visible when g_{i+1} has a critical event; otherwise, there is a contradiction in the sequence U giving the shortest path between p and q. □

Theorem 1: If R is simply connected and the robot is at a point in S, then the path encoded in the Gap Navigation Tree between the root and any point $q \in S$ is globally optimal in Euclidean distance.

Proof: Let $p \in S$ be the current position of the robot. From the GNT, a sequence $H = (g_1, g_2, ..., g_n, g_q)$ of gaps is generated such that if chased, the robot reaches q. By following H, the intervals $U = (u_1, u_2, ..., u_n)$ of the boundary are transversed by the robot, in that order. Let H_o be the sequence of gaps that generates the shortest path between p and q, as in Lemma 4. Let $g_d \in H$, generated from interval $u_d \in U$, be the first gap in which H and H_o differ. Since critical events are recorded in the GNT as they become visible, this means that the interval u_d is visible before the rest of the path in H_o, and it becomes visible when g_d splits. The shortest path between the current position of the robot and the rest of the path encoded in H_o is the one that starts by chasing g_d (by Lemma 4). Therefore, H_o contains g_d, and we conclude that $H = H_o$. □

Note that optimality follows uniquely from gap critical events, and no distance measurement is ever performed by the robot. For polygons, some of the intervals defined for U may degenerate into single points, and the gap-based roadmap corresponds to the *shortest-path roadmap*, also called the *reduced visibility graph* [25], [31]. It is important to remember that optimality is possible when the GNT is used for navigation, but the construction of the GNT may not be optimal. In fact, the distance traveled in the construction may be arbitrarily bad compared to the one traveled if the map of the environment was available [33]. If the environments have some restrictions, though, some bounds can be found for certain explorations, such as object searching inside *generalized streets*, presented in [7].

B. Complexity

In our case, the environment is unknown, and it is not *encoded* as an input to an algorithm in the usual sense. For this reason, we analyze the GNT complexity in terms of relevant

environment features. Consider the number n of inflections in the environment. The construction of the GNT cannot take more than $O(n)$ gap-chasing motion commands. Since there are $O(n^2)$ bitangents, and all of the visual events may be triggered while chasing a gap, the tree is updated at most $O(n^3)$ times. Note that this bound corresponds to the naive algorithm for constructing the visibility graph, in which each pair of vertices is tested for mutual visibility. The robot cannot predict a visibility event, given that R is unknown; thus, it has to sense each of the events to have all of the information of the shortest paths. When the GNT is completely constructed, its number of vertices is maximum when it is a complete binary tree, with a path to each of the inflections. Thus, it requires $O(n)$ space in the worst case. However, the GNT is not generally a complete binary tree, and it is not necessarily balanced. A query for $\mathbf{chase}(\cdot)$ takes in the worst case $O(n)$ time.

C. Traveling anywhere in the environment

Rather than be confined to the subset of R that corresponds to the gap-based roadmap, S, we would like to define tasks that allow the robot to move anywhere in R. It is difficult to even define such problems without using coordinates. A goal specified uniquely by its coordinates (i.e., (x, y)) does not have any meaning in the GNT framework. This is because the robot lacks any notion of coordinates. Suppose that the environment may contain both static, interesting places and some movable objects. Imagine that some objects are placed in R, and the robot is required to retrieve them. Let $O = \{o_1, o_2, \ldots, o_m\}$ be a collection of m point *objects*, and let $L = \{l_1, l_2, \ldots, l_n\} \subset R$ be a set of static *landmarks*. The robot could be asked, for example, to deliver objects from one landmark to another. Note that each landmark is a point in R. Each object has a current position in R at any given time; however, the precise position is unknown to the robot.

Assume that each object and landmark is uniquely identifiable and may be placed anywhere in R. An object $o_i \in O$ is said to be *recognized* when the robot is at $x \in R$ if and only if $o_i \in V(x)$ (that is, the object is inside the range of the visibility sensor). Recognition of a landmark is defined similarly. The gap sensor can be enhanced to recognize objects and landmarks. Let $G(x)$ be a cyclic sequence that may contain gaps or objects. If $o \in O$ and $o \in V(x)$, then $G(x)$ contains o precisely between the appropriate gaps from the robot's position. For example, if o_3 lies between gaps g_2 and g_5, then the sensor observation might be $G(x) = [g_1, g_2, o_3, g_5, g_7]$. Likewise, landmarks may also appear in $G(x)$.

For the task of retrieving objects or moving to landmarks, the motion primitive, chase$(\cdot)$, is adapted. A fifth critical event is included, which corresponds to the appearance of an object or landmark in $G(x)$. Thus, chase$(\cdot)$ terminates if a disappearance or split occurs to the gap being chased, or if an object or landmark appears. We also allow the robot to chase an object or landmark, yielding chase(o) or chase(l). To enable this primitive, the object or landmark must be visible from the robot position.

The algorithm from Section IV for constructing a complete GNT proceeds in the same way; however, additional information is now stored in the tree. The GNT definition is extended to allow objects and landmarks to appear as vertices. If an object or landmark disappears behind a gap g (it will look very much as a merge), then it is added to the GNT as a child of g (see Figure 9). The robot can return to any previously visible landmark l by chasing g until l appears in $G(x)$. We can therefore define $\mathbf{chase}(l)$, which is a sequence of motion primitive that leads to l. An object $o \in O$ can be handled in the same way in the GNT, resulting in $\mathbf{chase}(o)$. Objects are different from landmarks in that the robot is allowed to *carry* an object $o \in O$ to another part of R and drop it. As the robot moves away, o will be incorporated into the GNT in the appropriate way, in case a request is made to return to o.

The next theorem states that once completely constructed, the extended GNT can be used for navigation from the current position of the robot to any object or landmark in the environment.

Theorem 2: The extended GNT encodes a path to any object or landmark in the environment from the current position of the robot.

Proof: There are two cases for paths inside ∂R, depending of whether the object or landmark are visible from the current position of the robot. If it is visible, then the robot can travel in a straight line, following the line of sight. Otherwise, if the object or landmark was visible at some point, it is now hidden behind some gap, in which case following the sequence of gaps in the GNT will make it visible. We now prove that every point of R was visible at least once to the robot while constructing the GNT. Assume that there is at least one point, p, which was never visible. By definition, p is not currently visible, which means it is behind one of the currently detected gaps, say g (p either belongs to the current visibility region or not). If by chasing g it disappears, then p will be visible, which is a contradiction. If it splits, the argument is repeated recursively. Since all of the gaps are chased until they either split or disappear, all points of R are visible to the robot at least once. □

As a corollary to Theorem 1, we extend the path optimality for points not in S:

Corollary 1: In the extended GNT, $\mathbf{chase}(l)$ and $\mathbf{chase}(o)$ lead to distance optimal motions to l or o, between any possible pair of positions in R.

Proof: The argument is the same as the proof of Theorem 1. For paths starting in S, we only have to include the extension of the chase$(\cdot)$ motion primitive for objects and landmarks. For paths that do not start on S, consider the sequence of gaps that generate the optimal path, and the sequence read from the extended GNT. Apply the argument presented for Theorem 1 considering the gap for which they first disagree to be the first gap followed. □

VI. Multiply connected environments

So far we have only considered simply connected environments. Now we study the problem in which R is multiply connected, which is more common in practice. In this case ∂R

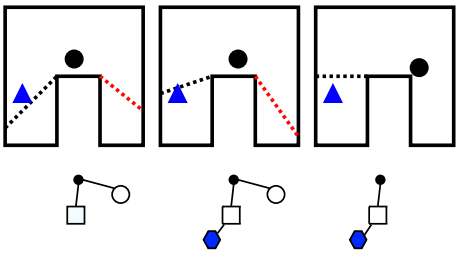


Fig. 9. Encoding objects in the Gap Navigation Tree. When the triangular object hides behind the gap, we associate such an object with the gap. The gap encodes the last time the object was visible (the object is *hidden* behind the gap).

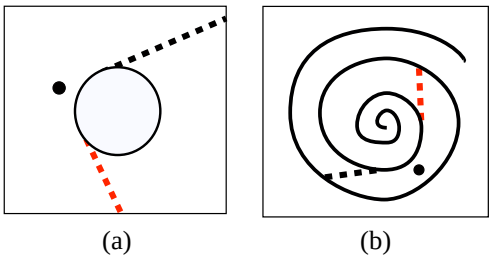


Fig. 10. Gaps in a multiply connected environment may not disappear. (a) The robot will chase any of the two gaps shown, and none of them will disappear. (b) Using only the gap sensor, the spiral looks the same as the disc in (a).

have several components. It is assumed that each component of ∂R is bounded and it is the image of a piecewise-analytic closed curve. The construction and use of the GNT was based on the motion primitive of chasing gaps until a critical event occurs. Although this offers a clean and simple feedback control to the robot, it is not sufficient for multiply connected environments. We state this negative result formally in the following theorem:

Theorem 3: Termination of gap-chasing motion primitives is not guaranteed in multiply connected environments.

Proof: Refer to Figure 10.(a). The environment does not have an inflection ray, or a bitangent complement. The robot will chase one of the gaps, expecting it to split or disappear, and it will keep going around a boundary component forever. □

Furthermore, only using gap sensing, path optimality in multiply connected environments cannot be achieved:

Theorem 4: Global path optimality is in general not possible using only gap sensing.

Proof: Consider Figure 11.(a). The robot has the choice of following a path to the left, or the right, to reach a goal. The path on the left is longer in the number of gaps to chase, but it is shorter in distance. The only information available is the number of gaps to chase, but this is only an indication of how "cluttered" a region is, and is not related to the distance to be traveled. Given that no length is associated to the gaps, both paths are equivalent, and the robot cannot determine which is shorter. Furthermore, as shown in Figure 11.(b), the path chosen by the robot may be arbitrarily longer than the shortest path. □

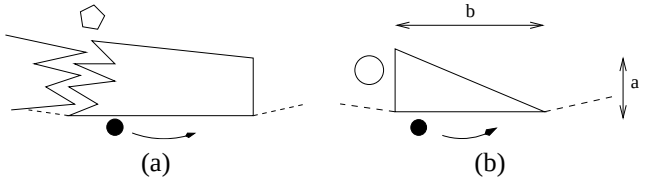


Fig. 11. (a) Global optimal navigation is not guaranteed in multiply connected environments. Paths with the least number of gaps are preferred. The robot will follow the path on the right, because it offers fewer gaps to chase, although it is not the shortest path. With only gap information, the robot cannot do better. (b) A worst case navigation example. If $b \gg a$, and if the robot chooses to follow the gap on the right, then almost the whole triangle boundary will be followed to reach the circle.

Together with these results, note that using only gap sensing a robot cannot determine whether it is surrounding a convex boundary component or it is traveling inside of an boundary component that has a spiral-like shape (compare Figure 10.(a) with Figure 10.(b)). Even though these negative results may be discouraging from an implementation point of view, they provide a clear formal distinction regarding minimal information requirements in simply and multiply connected environments. In simply connected environments, gap critical events are sufficient; in multiply connected environments they are not.

From a minimalist perspective, which critical events should be *added* for multiply connected environments? There are, of course, many ways in which this question can be answered, and it depends ultimately on the task the robot has to solve. In our case, we are interested in a data structure that encodes *at least* one path from the current position of the robot to any place in the environment. Thus, the data structure is still a tree. When two paths to the same location are detected in the GNT, the one with the least number of gaps is recorded, and the other one is eliminated. Although paths are no longer globally optimal, they are optimal in the homotopy class to which the path belongs. No algorithm based only on our gap-chasing model can do better. In [20], a similar problem is considered. It is solved by changing the direction of navigation if the robot moves in the direction opposite to the goal. Without a measurement of direction or distance, this is not possible under our gap-chasing model[2].

A. New assumptions for multiply connected environments

Some critical events should be introduced that guarantee a chasing motion primitive to terminate. One way to do this is to provide the robot with the capability of recognizing a location it visited before. This can be done providing the robot with *markers* or *pebbles*. When the robot makes contact for the first time with a boundary component while chasing a gap, a pebble can be dropped. Later, if by chasing the same gap, the pebble is found, a critical event is triggered. This indicates that the boundary component has been surrounded completely once. The robot is provided with a new motion primitive, surround(b), which commands the robot to transverse completely the boundary component b once.

[2]In practice, crude distance information could be used to make such decisions.

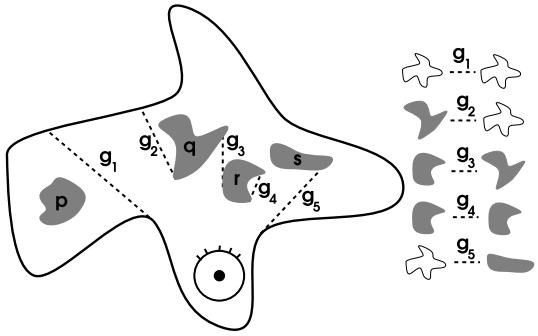


Fig. 12. Start and end of a gap. From the robot's perspective, the boundary components can be related to the beginning and end of a depth discontinuity.

The pebbles, as the gap sensor, are considered in an abstract sense, with an implementation that can vary. For example, they may be implemented using computer vision, or with a GPS (assuming that the error of the GPS is small in comparison to the size of the environment). Note that even if the GPS is available, its use is relegated exclusively to the pebble implementation. The particular implementation of the pebble is not important, as long as the robot is able to detect that it has surrounded a boundary component once. Note that the pebble *could be* implemented with a localization method, but localization itself is not required.

The number of pebbles needed depends on the particular GNT construction algorithm. As we will develop later, the algorithm proposed surrounds each boundary component, one by one, recording all the gap critical events. If we further assume that each of the boundary component is uniquely identifiable, only one pebble is needed. The somewhat strong assumption that each boundary component is identifiable is justified from a minimalist point of view. Such critical events should be detected to guarantee that the robot has explored the whole environment. As the size of the boundary components increases, the implementation of the sensors becomes more challenging. Nevertheless, such information should be present to make performance guarantees. In summary, the new requirements for multiply connected environments are a single pebble, and the ability to uniquely identify each boundary component.

As with objects and landmarks, components of ∂R are *associated* with a gap. Particularly, the start and end of the gap are associated with the respective boundary components. For example, in Figure 12, gap g_2 begins at boundary q and ends at the outer boundary. Gap g_3 begins at boundary component r and ends at boundary component q, and gap g_4 begins and ends at boundary component r. These are referred, respectively, as the *start* and *end* of a gap, and are recorded together with each gap, updating them accordingly if they change (a gap end may change without causing a visibility event). Note that if the boundary is never visible to the robot, there will not be a gap associated with it. This is the case when the boundary component cannot be reached in the connected component of R in which the robot is.

B. Constructing the GNT for multiply connected environments

To construct the GNT, the vertices are now classified into three types:

1) *Primitive:* Primitive vertices encode gaps that appear as the robot moves.
2) *Nonprimitive:* Nonprimitive vertices are the parents of vertices corresponding to gaps that merge, or they are leaves that are not primitive but that merged with an object, a landmark, or they are the only gap associated with a particular boundary component.
3) *Block:* A block vertex is a leaf that is not primitive, and its associated boundary components are the same of some primitive or nonprimitive gap.

As before, when the robot is placed in a new environment, all of the leaves of the GNT are marked nonprimitive because the robot has not yet seen what is behind the corresponding gaps. To guarantee that the robot will see the whole environment, a surround motion primitive is executed for each of the components of ∂R. The robot chooses arbitrarily to follow a boundary component not traversed before, and once this is completed, a new boundary component is selected. Incrementally, the robot determines how to reach every boundary component, as various gaps get associated with them.

The critical events are encoded in the same way as before, with the following exception. Since the environment is multiply connected, the homotopy class of paths between two locations may not be unique. From the robot's perspective, however, all paths through the GNT are equivalent because the robot lacks distance information. Therefore, paths that chase the least number of gaps are preferred. This heuristic does not guarantee that optimal paths will be preserved, but prefers paths that drive the robot through less cluttered areas. To achieve this, some paths are eliminated from the GNT as follows. Suppose that the sequence $[g_1, ..., g_m]$ of gaps read from the GNT is the shortest sequence of gaps which reach a particular object, landmark, or boundary component. The association of the object, landmark or boundary component with any gap other than g_m is removed. If the object, landmark or boundary component is visible from the current position of the robot, then the association with any vertex is removed.

Given the previous procedure, some nonprimitive gaps may not be associated with any object, landmark or boundary component. Any vertex corresponding to such gap is labeled as *block*. Furthermore, if all of the children of a vertex are labeled as block, and the vertex itself is not associated with an object, landmark or boundary component, then the vertex itself is labeled as block and all of its children are eliminated. Thus, two block vertices cannot merge in the tree, since only one is kept. Figure 13 illustrates this process. A splitting block vertex yields two block vertices. A block vertex returns to a nonprimitive status if it is associated with a new goal. In a sense, keeping a branch full of block vertices does not increase the robot knowledge of the environment, because it can reach all of the goals chasing other gaps. To summarize the previous discussion, the following theorem is presented:

Theorem 5: In the GNT for multiply connected environments $\mathbf{chase}(l)$ and $\mathbf{chase}(o)$ lead to motions locally optimal

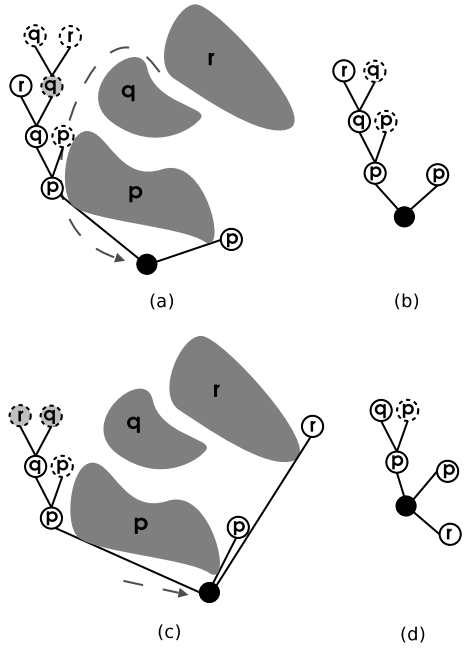


Fig. 13. Block vertices elimination. The vertices in the trees are marked with the corresponding boundary component at their starts. Block vertices are denoted with dotted outlines. In (a), after the robot transverses the trajectory shown, the shaded vertex will be eliminated, since itself and its children have goals already in the tree, yielding (b). As the robot moves (c), the boundary component r becomes associated with a gap closer to the root, which produces further elimination of block vertices (d).

in distance to l or o, between any possible pair of positions in R.

Proof: With the argument of the proof for Theorem 2, we argue that each object, landmark and boundary component is visible at least once to the robot. Thus, if vertices are not eliminated from the GNT, there is at least one path to each object, landmark, or boundary component. Furthermore, vertices are eliminated only if there is another path (which generates a smaller sequence of gaps). Once such path is selected to remain in the GNT, local optimality follows from Theorem 1. □

VII. Implementation

We implemented a simulation of the algorithms for the GNT in simply and multiply connected environments. We also validated the GNT sensing requirements on a mobile robot.

A. Simulations

Figure 14 shows a simulation of the GNT construction. The position of the robot is marked with the large black disc, which also serves as the root of the graphical representation of the GNT. Primitive vertices are shown with a square; nonprimitive ones with a disc. Dark (red) and light (green) color vertices hide regions of the environment to the right or to the left, respectively. The branches of the GNT are aligned with the gap they represent, but this is only for clarity of presentation. Recall that no exact angular information is used when constructing the GNT. The initial position of the robot is shown in Figure 14.(a). Because no gap has been explored, all vertices of the tree are circles (nonprimitive gaps). Figure 14.(b) and (c) show different states of the the GNT, as nonprimitive gaps are being explored. In Figure 14.(c) it is particularly clear that the two vertices on the bottom-left are encoding two regions that will not split, whereas all of the other branches encode all of the other possible destinations in the environment. Figure 14.(d) shows the moment just before the last nonprimitive gap disappears. It can be seen that all other leaves of the GNT are squares.

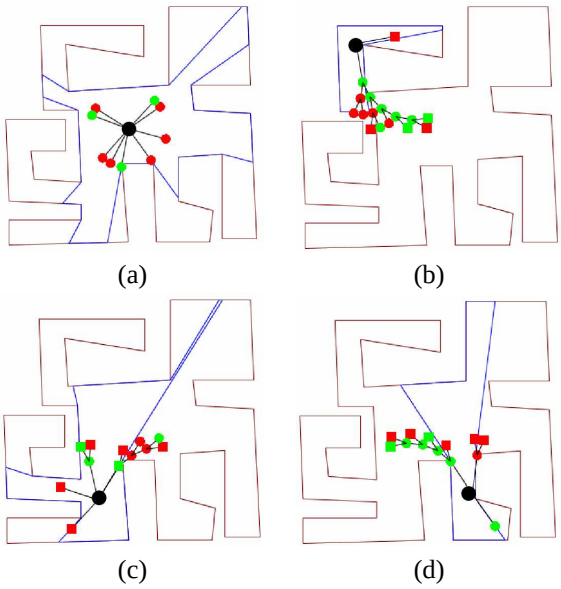


Fig. 14. Graph Navigation Tree simulation for a simply connected environment. (a) Initial position. (b) and (c) Intermediate moments in the construction. (d) The instant before the last nonprimitive gap disappears.

Figure 15 shows an example in which the robot was asked to construct the GNT, and to encode the position of the objects present in the environment. Once this task is completed, the robot moves all the square objects of a certain color to the corresponding circle of the same color. Figure 15.(a) shows the tree when the construction is completed (all of the leaves are squares). From this point, the robot begins to deliver the objects until this new task is completed (Figure 15.(b)).

Finally, in Figure 16, a simulation in a multiply connected environment is presented. The GNT is shown for the position of the robot, denoted as a black disc. The two trees correspond to the GNT before and after the boundary component b_3 was surrounded.

B. Experiments with a Pioneer mobile robot

We performed some limited experiments to test the validity of our sensing model. The platform was a Pioneer 2-DX with two laser range finders. A merge event is shown in Figure 17, in which two gaps are merged into one when the robot hides behind a corner. Although the gap-chasing procedure implemented here was enough for our simple experiments, a more robust navigation system following discontinuities should be implemented (such as the one presented in [30], which uses similar models).

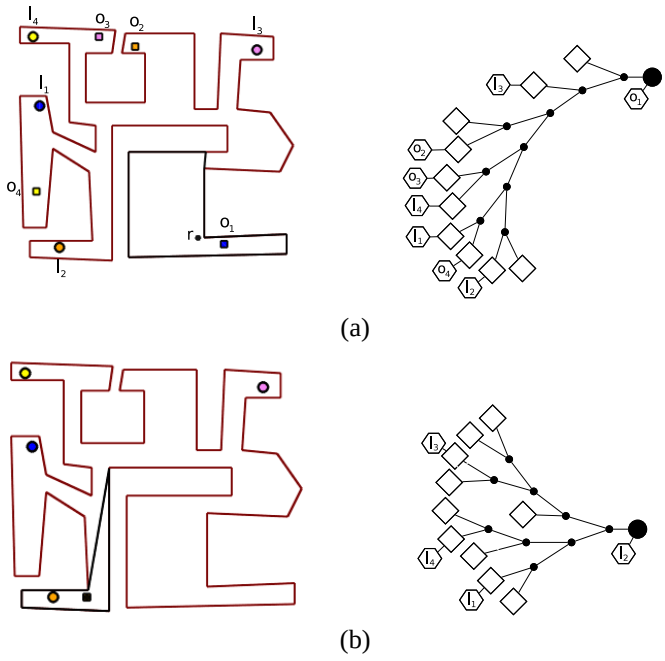


Fig. 15. Object finding simulation for a simply connected environment. In (a) the state of the Gap Navigation Tree is shown, after the construction phase concluded. In (b) the tree is shown, after all of the objects have been delivered.

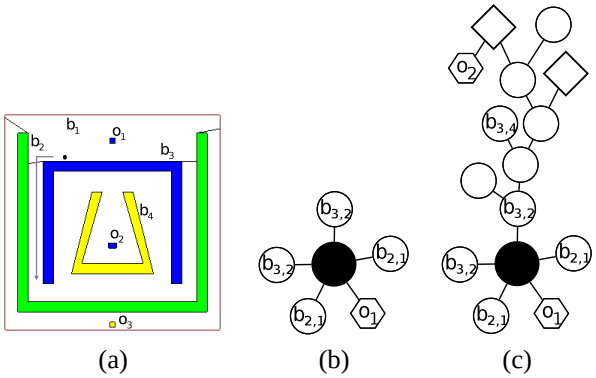


Fig. 16. Simulation in a multiply connected environment. The GNT is shown for the position of the robot denoted with the the black disc in (a). In (b) and (c), the GNT is shown before and after the boundary component b_3 was surrounded, respectively. The label b_{ij} of some vertices indicates their start and end boundary component. Vertices without such label are either primitive or block.

Even though our experiments were much simpler than the ones we used in simulation, we are hopeful that the sensing requirements are met in many real settings. For example, maps also based on visibility events and constructed by two robots were presented in [36] for more challenging environments. Further experimental work is still needed to evaluate the full applicability of our model. For example, the use of two laser range finders seems an overkill for the problem at hand. Also, the experiments were done in an artificial environment using cardboard as the environment boundary. The robustness of gap sensing in the presence of *small gaps* in a real environment remains to be tested. Finally, and perhaps more importantly, in a real setting the robot is not a point. This raises two main issues. The first is that a robust wall-following

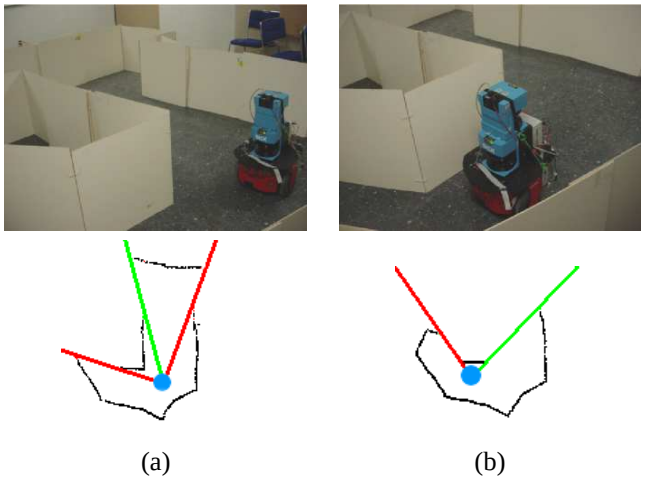


Fig. 17. Gaps merging. Figures (a) and (b) show the position of the robot and the corresponding gaps before and after the merge critical event is detected.

procedure should be implemented (this becomes even more important in the multiply connected case). The second issue is that optimality is potentially lost, because the gap-chasing procedure does not incorporate the robot radius.

VIII. Conclusions

This paper presented the Gap Navigation Tree (GNT) data structure, and its use for optimal navigation in unknown environments with a robot that has limited sensing. The navigation paths are optimal, even though no distance or other exact geometric measurements are made by the robot. The GNT is constructed from detecting online visual events, yielding a simple but powerful framework for solving visibility based tasks in the plane. Our approach studies minimal information a robot should gather to solve a task. The GNT shows that the need of an exact geometric representation of the environment, or localization, may be ignored to solve some visibility-based tasks. Moreover, not only are exact representations eliminated, but the information needed by the robot can be greatly minimized.

The GNT is well suited for solving other visibility tasks. We applied it for pursuit-evasion in unknown environments [19], but its use for other tasks is still open. As mentioned in the introduction, the localization problem is easily described in terms of visual events, and the use of the GNT in this case is straightforward, following the algorithms in [11], [18][3].

One common concern from an experimental point of view is that range data can be used to make gap tracking more robust. Moreover, it may seem strange that we discard such data if it is readily available. Our paper, however, is concerned with determining the least information needed by any algorithm. In practice, however, extra information could be used to increase robustness. The ideal gap sensor as presented here would likely fail in a real setting due to noise. The range information may be used to construct a better gap sensor, but in the end, only

[3]Instead of comparing *visibility skeletons*, configurations of the GNT should be used. It is remarkable that such an algorithm will work with exactly the information discarded by the visibility skeletons, because the spurious edges correspond to the visible gaps in an aspect cell.

the gaps matter. Although experiments validating the sensing model were presented, some practical issues remain to be solved. The most important ones are related to the gap tracking process. Since the gaps have to be tracked at all times, the implementation of the gap sensor should be robust enough. An important issue is when a critical event is not detected by the robot (due a noisy reading from the gap sensor, for example). In this case, complete branches of the GNT may be lost, or the robot may chase the "wrong" gaps to reach a goal. An interesting problem is to detect such errors also from the critical events, and to devise a strategy to recover as much of the already-built tree as possible. Another direction is to determine which sensing capabilities should be added to make the gap sensor more robust. In doing so, the minimalist approach should be still considered in terms of the abstract sensors. Our proposal here is that by removing information requirements in the algorithmic side, measurement errors in the sensors could be solved cleanly and directly. One example of this is the pebble implementation proposed in Section III, through a GPS. If a GPS is available, it is tempting to build a geometric model of the environment, since the localization problem has been solved for us. In such a case, we will have to rely on the error measurements of the range finders, and of the GPS itself. If instead the effort focuses on the task at hand, we may find that such measurements are not needed. In the tasks presented in this paper, only depth discontinuities, and a pebble should be detected. Using the GPS as a pebble solves the sensing requirement easily and cleanly, and we may even ignore the GPS measurement errors. Moreover, the sensing requirement is not specified as a particular implementation. The pebble can be implemented through computer vision, dropping RFID tags [22], or even the unlikely scenario of a robot arm dropping *real* pebbles.

ACKNOWLEDGMENTS

The authors thank Javier Minguez, Boris Simov, Stjepan Rajko, Shai Sachs and Wes Huang for useful discussions and suggestions. The robot platform 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*, 1533:357–366, 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 and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[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. Bulata and Michel Devy. Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot. In *IEEE Int. Conf. Robot. & Autom.*, pages 1054–1060, 1996.

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

[7] A. Datta and C. Icking. Competitive searching in a generalized street. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 175–182, Stony Brook, New York, 1994.

[8] G. Dedeoglu, M. J. Mataric, and G. S. Sukhatme. Incremental, on-line topological map building with a mobile robot. In *Mobile Robots XIV - SPIE*, pages 129–139, 1999.

[9] B. R. Donald. On information invariants in robotics. *Artif. Intell.*, 72:217–304, 1995.

[10] G. Dudek, P. Freedman, and S. Hadjres. Using local information in a non-local way for mapping graph-like worlds. In *Proc. Int. Joint Conf. on Artif. Intell.*, pages 1639–1645, 1993.

[11] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 437–446, 1995.

[12] F. Durand. *3D Visibility: Analytical study and applications*. PhD thesis, Université Grenoble I – Joseph Fourier Sciences et Géographe, July 1999.

[13] A. Elfes. Sonar-based real world mapping and navigation. *IEEE J. Robot. & Autom.*, 3(3):249 – 264, 1987.

[14] M. Erdmann. Understanding action and sensing by designing action-based sensors. *Int. J. Robot. Res.*, 14(5):483–509, 1995.

[15] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Trans. Robot. & Autom.*, 4(4):369–379, August 1988.

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

[17] H. H. Gonzalez, E. Mao, J. C. Latombe, and T. M. Murali. Planning robot motion strategies for efficient model construction. In *Robotics Research - The 9th Int. Symp.*, pages 345–352, 1999.

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

[19] 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*, volume 4, pages 3456– 3462, 2004.

[20] I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. *IEEE Trans. Robot. & Autom.*, 13(6):814–822, December 1997.

[21] I. Kamon, E. Rivlin, and E. Rimon. A new range-sensor based globally convergent navigation algorithm for mobile robots. In *IEEE Int. Conf. Robot. & Autom.*, 1996.

[22] A. Kleiner, J. Prediger, and B. Nebel. RFID technology-based exploration and SLAM for search and rescue. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 4054–4059, Beijing, China, 2006.

[23] J.J. Koenderink. *Solid Shape*. MIT Press, 1990.

[24] J.J. Koenderink and A.J. van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, 24:51–59, 1976.

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

[26] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. Available at http://msl.cs.uiuc.edu/planning/.

[27] 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.

[28] 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.

[29] L. Matthies and A. Elfes. Integration of sonar and stereo range data using a grid-based representation. In *IEEE Int. Conf. Robot. & Autom.*, pages 727–733, 1988.

[30] J. Minguez and L. Montano. Nearness diagram navigation (ND): Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20:45–59, February 2004.

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

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

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

[34] M. Pocchiola and G. Vegter. The visibility complex. *Int. J. Comput. Geom. & Appl.*, 6(3):279–308, 1996.

[35] 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.

[36] Ioannis M. Rekleitis, Gregory Dudek, and Evangelos Milios. Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):7–40, 2001.

[37] E. Remolina and B. Kuipers. Towards a general theory of topological maps. *Artificial Intelligence*, 152:47–104, 2004.
[38] S. Sachs, S. Rajko, and S. M. LaValle. Visibility-based pursuit-evasion in an unknown planar environment. *To appear in International Journal of Robotics Research*, 23:3–26, 2003.
[39] H. Shatkay and L. P. Kaelbling. Learning topological maps with weak local odometric information. In *International Joint Conference on Artificial Intelligence*, pages 920 – 929, 1997.
[40] S. Thrun, W. Burgard, and D. Fox. Probabilistic mapping of an environment by a mobile robot. In *IEEE Int. Conf. Robot & Autom.*, pages 3–26, 1998.
[41] 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*, pages 11–26, 2004.
[42] 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.
[43] 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 3491–3497, 2003.

Benjamin Tovar is currently a Ph.D. student in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He received the B.S. degree in electrical engineering from the ITESM Campus Mexico City (2000), and the M.S. in electrical engineering from the University of Illinois at Urbana-Champaign (2004). His research interests include minimal sensing for robotics, motion planning, planning algorithms, control theory and computational geometry.

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 an Associate Professor in the Department of Computer Science at the University of Illinois. His research interests include planning algorithms, motion planning, computational geometry, and control theory. He authored the book Planning Algorithms, Cambridge University Press, 2006 (which is available on line for free).

Rafael Murrieta-Cid received his Ph.D. from the Institut National Polytechnique (INP) of Toulouse, France (1998). His Ph.D research was done in the Robotics and Artificial Intelligence group of the LAAS/CNRS. In 1998-1999, he was a postdoctoral researcher in the Computer Science Department at Stanford University. From January 2000 to July 2002 he was an assistant professor in the Electrical Engineering Department at Tec de Monterrey Campus México City, México. In 2002-2004, he was working as a postdoctoral research associate in the Beckman Institute and Department of Electrical and Computer Engineering of the University of Illinois at Urbana-Champaign. From August 2004 to January 2006 he was professor and director of the Mechatronics Research Center in Tec de Monterrey, Campus Estado de México, México. Since March 2006, he has been working in the Mathematical Computing Group at the CIMAT – Centro de Investigación en Matemáticas, in Guanajuato México. He is mainly interested in sensor-based robot motion planning and computer vision.