Visibility-Based Pursuit-Evasion in an Unknown Planar Environment

Shai Sachs
Dept. of Computer Science
University of Illinois
Urbana, IL 61801 USA
ssachs@uiuc.edu

Stjepan Rajko
Dept. of Computer Science
Iowa State University
Ames, IA 50011 USA
stipe@iastate.edu

Steven M. LaValle
Dept. of Computer Science
University of Illinois
Urbana, IL 61801 USA
lavalle@uiuc.edu

Abstract

We address an on-line version of the visibility-based pursuit-evasion problem. We take a minimalist approach in modeling the capabilities of a pursuer robot. A point pursuer moves in an unknown, simply-connected, piecewise-smooth planar environment, and is given the task of locating any unpredictable, moving evaders that have unbounded speed. The evaders are assumed to be points that move continuously. To solve the problem, the pursuer must for each target have an unobstructed view of it at some time during execution. The pursuer is equipped with a range sensor that measures the direction of depth discontinuities, but cannot provide precise depth measurements. All pursuer control is specified either in terms of this sensor or wall-following movements. The pursuer does not have localization capability or perfect control. We present a complete algorithm that enables the limited pursuer to clear the same environments that a pursuer with a complete map, perfect localization, and perfect control can clear (under certain general position assumptions). Theoretical guarantees that the evaders will be found are provided. The resulting algorithm to compute this strategy has been implemented in simulation. Results are shown for several examples. The approach is efficient and simple enough to be useful towards the development of real robot systems that perform visual searching.

1 Introduction

In the past few years, there has been significant interest in robotics and computational geometry in designing motion strategies for pursuit-evasion scenarios. The basic task is to move one or more robots (pursuers) to guarantee that unpredictable targets (evaders) will be detected using visibility-based sensors. Efficient algorithms that compute these strategies can be embedded in a variety of robotic systems to locate other robots and people. Potential application areas include surveillance, high-risk military operations, video game design, search-and-rescue efforts, firefighting, and law enforcement.

1.1 Relation to Previous Work

Previous work has considered the case of having a complete map and a robot with perfect navigation capabilities; this case is already quite challenging. Visibility-based pursuit-evasion in a polygonal environment was first considered in [30], in which algorithms were given for classes of polygons in which the pursuer has omnidirectional visibility, or has a set of k beams, called flashlights. A complete algorithm for the case of omnidirectional visibility was first presented in [20]. Solutions to the case in which the pursuer has one or more detection beams, or in which several pursuers are present, are considered in [7, 21, 28, 33, 31], for various types of polygons. A pursuit-evasion algorithm for curved environments was presented in [19]. In [11], both optimal and approximation algorithms were presented for the case of a chain of pursuers that maintain mutual pairwise visibility in a polygonal environment. Until very recently [25], it had been unknown for

nearly a decade whether the omnidirectional pursuit-evasion problem in a polygonal environment could be solved in polynomial time (it can be solved in quadratic time).

This paper presents the first approach to visibility-based pursuit-evasion in a curved environment that does not require a complete map of the environment to be specified *a priori*. Our work follows on-line and minimalist approaches both in theoretical algorithms [5, 8, 24] and in robotics [1, 2, 3, 6, 9, 10, 12, 13, 29, 17, 18, 23, 22]. Our models are similar in some ways to those presented in [23] for robot navigation. In that work, *bug algorithms* were presented for navigating a robot to a goal, in the absence of a map and with very limited sensing. The robot is assumed to have very simple capabilities, such as contact sensing and wall following; yet, it is guaranteed to reach its destination efficiently. In recent years, other navigation algorithms have been designed using this philosophy [6, 16, 17, 27]. This mindset inspires our current work, which applies these principles to pursuit-evasion problems, resulting in a kind of pursuit-evasion bug algorithm. The primary benefit of this approach is that the sensing requirements of the robot are greatly reduced.

The pursuit-evasion problem, however, is considerably more challenging than basic navigation. Imagine arriving in a dark, unfamiliar environment with a lantern, and trying to search for any people who might be moving around. It is assumed that you have no map, no compass, and no reliable way to measure distances needed to construct a reliable geometric map. This scenario is similar to what might be confronted by a robot to accomplish tasks such as search-and-rescue. In many applications, it is too risky, expensive, or unreliable to construct an "exact" map of the environment. Robot sensors are costly and usually inaccurate. Robotic systems that rely heavily on the integration of many sensors are often prone to error. Furthermore, map building is a time consuming process that is complicated by issues such as segmentation, viewpoint registration, and outlier suppression. Once a map is obtained, precise localization within the map becomes the next obstacle, which is a challenging problem that has been considered extensively in experimental mobile robotics research. Due to common problems such as poor odometry and wheel slippage, localization must be performed frequently to determine the appropriate movement with respect to the map. By taking a minimalist approach in this context, we can circumvent many problems that arise in experimental robotics; this approach can lead to low-cost robot systems that solve tasks with high reliability.

Therefore, we depart from the traditional assumption of perfect mapping and navigation capabilities. We introduce an on-line version of the problem by defining simple sensing requirements in the model, and requiring that all motions are executed directly with respect to the sensor readings. We present an algorithm which, under general position assumptions, yields a on-line motion strategy that is guaranteed to find any unpredictable evaders, if such a strategy exists for the limited pursuer.[1] Furthermore, we prove that the algorithm yields a pursuer with searching power that is equivalent to a pursuer with a complete map and perfect localization. This result is significant, because we assume a very limited pursuer model, and because the task is extremely difficult, even given a better-equipped pursuer (such as one with a map of R).

A similar approach was used in [29]. In that work, an on-line algorithm was given for a pursuer whose visibility is limited to a single flashlight, and whose motion is confined to the boundary of a polygonal region. It was proven that this searcher could clear the same environments that a pursuer with 360 degree vision, confined to motion on the boundary, could clear.

1.2 An Overview of the Approach

The case of pursuit-evasion in an unknown environment is substantially more complicated than the case of pursuit-evasion in a known environment. Therefore, we provide a high-level overview of the ideas contained in this paper, to help give motivation and an intuitive understanding of the concepts before proceeding to the details in the coming sections.

[1] A preliminary form of this work appeared in [26].

The key components are as follows:

1. **Gap sensing and control** Section 2 presents a model which assumes that the pursuer can use a sensor to determine the directions of discontinuities in depth (resulting in a *gap sensor*) as observed from the current pursuer location. The pursuer is restricted to a simple set of motion primitives that are either expressed entirely in terms of the gaps or in terms of wall following. This model enables a robot to solve pursuit-evasion problems without being able to measure precise depths, build maps, read a compass, use odometry, or localize itself in any coordinate frame. In related work [32], this model was validated through experiments performed in an indoor environment on a Pioneer 2-DX mobile robot equipped with two SICK lasers.

2. **The navigation graph** Section 2.4 defines a set of carefully-chosen motion primitives. Each motion primitive guides the pursuer within a thin strip in its environment, leading from one set of possible locations to another set of possible locations. From the perspective of the pursuer, these sets of possible locations are simply abstract states. Each application of a motion primitive leads to an edge between the corresponding abstract states, resulting in a *navigation graph*. Although the map is not known to the pursuer, for any environment in which the robot is placed, the arrangement of the thin strips leads to a planar graph (Theorem 4.9). The motion commands are defined in such a way that the strips are actually "thickened" bitangent rays, as defined in [19].

3. **The status graph** The information state (referred to here as the *status*) must be maintained at all times by the pursuer, just as in [14, 19]. In previous work, this involved keeping track of when the pursuer crosses from one information-conservative cell to another, and appropriately updating binary labels. Each label corresponds to a connected component of the environment that is not visible, and indicates whether or not an evader may be in that component. Using the gap sensor model, this means that a binary label is assigned to each gap. Initially, all gaps are labeled to indicate that evaders could be in any component that is not visible. The goal is to force all labels to indicate that there are no hiding evaders. As the pursuer moves, labels must be updated correctly, which will be described in Section 2.5. This is an on-line analog of the appear/disapper/split/merge cases that appeared in [19]. Note that the pursuer may return to the same state in the navigation graph several times, but each time it could have a different status. Intuitively, it is like having a fiber over each vertex in the navigation graph, which yields a set of information states. This results in a directed *status graph*, which is analogous to the information state graph in [14, 19], but applies to the on-line case. See Section 3 for details.

4. **Envisioning** Solving the pursuit-evasion problem ultimately involves exploring the status graph. The concept of *envisioning* arises in the on-line version of the pursuit-evasion problem, but does have a counterpart for the perfect mapping and control case. In the perfect case, the pursuer can "envision" a solution (i.e., search the status graph) without having to move through the environment. Its motion strategy does not actually depend on anything learned during execution. In the on-line case, the pursuer learns information about the environment, and incrementally constructs the navigation graph. At any point, it can "stop and think" about what would happen if it were to move in different ways through the navigation graph that it built so far. This corresponds to searching the portion of the status graph that is known so far, which we refer to as *envisioning*. One approach would be to learn the entire navigation graph and then search the resulting status graph. Our approach is to tightly interleave exploration with envisioning so that the pursuer can solve a pursuer-evasion problem without necessarily having the entire navigation graph. These issues are discussed in Section 3.

5. **Touring countries** In previous work [19], a known environment was divided into cells based on rays that were extended outward along bitangents and also on rays extended from inflectional tangents. The resulting arrangement of cells can be considered as an aspect graph based on perspective projection. Imagine one cell decomposition based only on bitangents, in which each 2D cell is referred to as a "country". Imagine a refined cell decomposition in which both bitangents and inflections are used, and refer to each resulting cell as a "province". Thus, each country is partitioned into provinces by inflection rays.

   The key intuition behind the movement strategy is that unnecessarily crossing country borders (i.e. bitangent rays) can lead to information loss because connected components of the invisible portion of the environment could become merged. Furthermore, by moving the pursuer along the navigation graph, the pursuer only stays within the "border region" of any country. It has to be argued that there is no reason to visit interior provinces in any country (Theorem 4.4); all information regarding the pursuit can be obtained by traveling along the border region. In addition to this difficulty, one must travel from country to country in a way that avoids unnecessary loss of information.

6. **Recognizability problems** A recurring difficulty throughout this work is the challenge of being able to recognize the occurrence of specific events, even though basic sensors, such as odometry and a compass, are missing. For example, when traveling from country to country, how can we tell whether we are returning to a country already visited, or arriving in a new country for the first time? This is addressed by selecting a careful movement strategy, presented in Section 3. Following this, recognizability arguments form a large part of the algorithm analysis presented in Section 4. When traveling within a country, how do we know that we have traversed the entire boundary at least once? This problem is more difficult to solve, and we simply assume some additional capability on the part of the pursuer, such as being able to place a marker on the floor and being able to recognize upon return. In practice, it should be easier to detect and avoid such infinite loops, but the theoretical technicality nevertheless exists.

7. **Completeness** Section 4 covers the arduous task of showing that the algorithm presented in Section 3 enables the pursuer to solve pursuit-evasion problems for as many environments as it possibly can. In fact, under general position assumptions, it can even solve any pursuit-evasion problem that can be solved by the original pursuer with perfect mapping and control [14, 19, 30]. This is stated in Theorem 4.12, which concludes the theoretical analysis. Thus, the restrictive model developed in this paper surprisingly leads to a pursuer that is just as powerful as the original pursuer. Note, however, this will come at a price: the distance traveled by the pursuer presented in this paper is much greater.

An implementation (in simulation) is briefly presented in Section 5, along with some computed examples. Finally, some conclusions and final remarks appear in Section 6.

2 The Model

2.1 Problem Definition

The task is for one *pursuer* robot to visually locate one or more *evaders* in a complicated environment. The pursuer and evaders are each points that move in a bounded, simply-connected open set, R, in the plane. The boundary of R is a simple, closed, piecewise-smooth curve (with only a finite number of nonsmooth points). Note that this definition includes polygonal boundaries. Let $\pi_{e_i}(t) \in R$ denote the position of the i^{th} evader at time $t \geq 0$. It is assumed that $\pi_{e_i} : [0, \infty) \rightarrow R$ is a continuous function, and each evader is

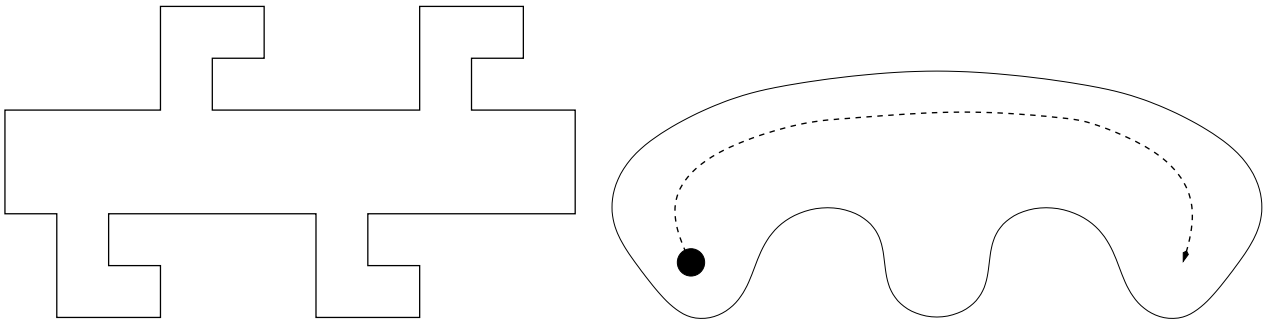


Figure 1: a) An example of a region which cannot be cleared. b) An example of a region which can be cleared, and one path which clears the region.

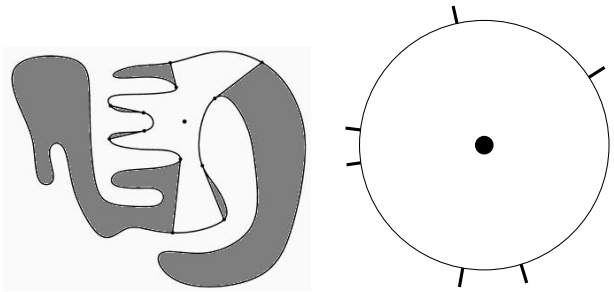


Figure 2: Discontinuities in depth measurements partition the set of viewing directions. Each discontinuity could hide an evader.

capable of moving arbitrarily fast (i.e., it moves at a finite, unbounded speed). Similarly, let $\pi_p(t)$ denote the position of the pursuer at time $t \geq 0$; the pursuer also moves continuously.

For any $x \in R$, let $V(x) \subseteq R$ denote the set of all $y \in R$ such that the line segment that joins x and y does not intersect the boundary of R. We call $V(x)$ the *visibility region* of x. If at any time t, $\pi_{e_i}(t) \in V(\pi_p(t))$, we say that the i^{th} evader is detected.

For convenience, we will assume that an evader is eliminated when it is detected. Any such subset of R that might contain a evader is referred to as a *contaminated* region. If it is guaranteed not to contain any evaders, then it is referred to as *cleared*. If a region is contaminated, becomes cleared, and then becomes contaminated again, it is referred to as *recontaminated*.

It is assumed that the pursuer does not know the starting position, $\pi_{e_i}(0)$, or the path, π_{e_i}, of any evader, e_i. Initially, each evader could be anywhere in R that is not visible from $\pi_p(0)$. The task is to ensure that the pursuer will follow a path that clears R, or to report that it is impossible to clear R. See Figure 1.

Finally, our general position assumption is that no three *critical lines* intersect at a single point, in which a critical line is either a generalized inflection of ∂R or a generalized bitangent to ∂R, as defined in [19]. For the case of curved environments, these reduce to inflections and bitangents. The generalized definitions also encompass equivalents for polygonal environments. We also assume that there are no triple tangents. Note that the general position assumptions are defined for the environment in which the robot is placed, even though the robot does not have a map of R.

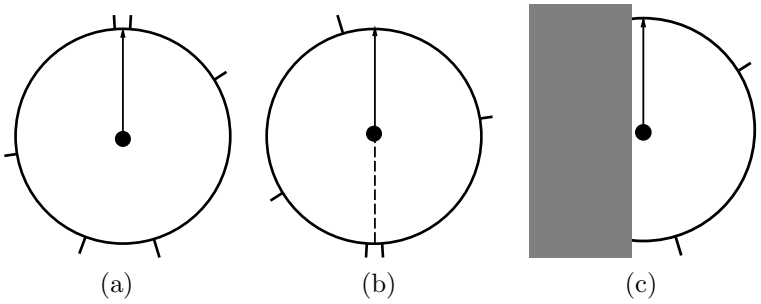


Figure 3: Three situations: a) moving towards a proximity pair; b) moving away from a proximity pair; c) moving along a wall.

2.2 Pursuer Model

The pursuer robot has no representation of R initially, and is not capable of building an exact, complete map of its environment. It is assumed that the pursuer's sensors are not necessarily powerful enough to provide accurate and dense depth measurements, which could be used to build a representation and perform localization. In the interest of robustness, it might be advantageous to avoid such constructions even if the pursuer is capable. It is assumed that the pursuer can execute omnidirectional motions, but with limited precision that results in control uncertainties. The pursuer has neither a reliable odometer nor a compass.

Some sensing capability is required, of course, to solve the problem. Consider Figure 2.a, in which a pursuer is placed in a curved environment. The pursuer is equipped with a sensor that is capable of producing a representation as shown in Figure 2.b, which gives the location of discontinuities in depth information, when performing an angular sweep of the environment. Note that each discontinuity corresponds to a connected portion of R that is not visible to the pursuer. Each discontinuity will be referred to as a *gap*, and the sensor will be called the *gap sensor*. It is assumed that the pursuer can track the gaps at all times, and record any topological change, which involves the appearance, disappearance, merging, or splitting of gaps.

2.3 Gap Tracking

As the pursuer moves, the report from the gap sensor will change. To describe the changes which may occur and the events which cause them, we define two types of lines in R, as shown in Figure 4. Consider a ray which extends from an inflection of the boundary until it hits another point on the boundary; we refer to this ray as an *appear line*. Also, consider a line segment tangent to the boundary at two points (that is, a bitangent). A ray can be extended outward from each point of tangency until it hits the boundary; we refer to each ray as a distinct *merge line*. Note that there are two merge lines for every bitangent.

There are four possible ways in which the gaps change. Below, we describe each event and provide notation for the event, assuming that the event happens to a gap γ. See Figure 5.

1. A new gap appears: $appears(\gamma)$
2. An existing gap disappears: $disappears(\gamma)$
3. Two or more gaps merge into one: $\alpha = merge(\beta, \gamma)$
4. A single gap splits into two or more gaps: $split(\gamma) = (\alpha, \beta)$

We call these changes *gap events*. The first two events occur because the pursuer crosses an appear line. The other two events occur because the pursuer crosses a merge line [19]. Suppose that γ splits when the

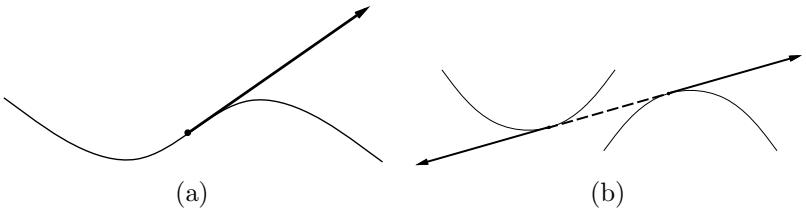


Figure 4: Rays extended from inflections and bitangents cause critical events. The rays are terminated at their first point of intersection with the boundary. a) the ray represents an appear line. b) the two rays each represent a distinct merge line.

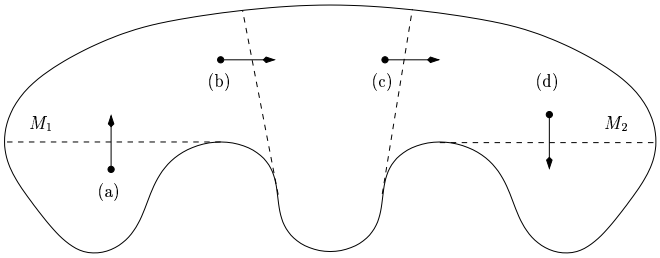


Figure 5: Four critical events: a) a single gap splits into two gaps; b) a gap disappears; c) a gap appears; d) two gaps merge into a single gap. Appear lines and merge lines are indicated by dashed lines.

pursuer crosses some merge line M. We refer to the subset of R in which γ is split as the *split side* of M, and we denote it $R_{s(M)}$. The other side is referred to as the *merged side* of M, and is denoted $R - R_{s(M)}$. In Figure 5(a), the robot moves from $R - R_{s(M_1)}$ to $R_{s(M_1)}$, and in Figure 5(d), the robot moves from $R_{s(M_2)}$ to $R - R_{s(M_2)}$.

If the robot follows a path $\pi : [t_0, t_1] \rightarrow R$ and a gap γ is not involved in any gap events while the robot moves, then we assume that at any moment during execution of π, the robot can determine which of the gaps reported by the gap sensor is γ, even if its angle changes. If γ is involved in a gap event, then we assume the robot can detect the gap event, and can also determine the other gaps affected, both before and after the event occurred. These abilities are referred to as *gap tracking*.

We next specify the control precision required for the robot. Suppose that the robot is commanded to follow a merge line or to follow part of the boundary of R. Let L denote the merge line or the part of the boundary that is commanded. Let ϵ be the maximum distance of the robot from L as it executes this command. Consider the arrangement of appear lines, merge lines, and walls in R. Form a second arrangement by making two replacements: 1) each merge line M is replaced by a parallel lines that is distance ϵ from the merge line, in $R_{s(M)}$; 2) the walls are replaced by curves that are distance ϵ from the walls, in R. It is assumed that ϵ is small enough that the arrangement formed by the replacements is equivalent to the first arrangement.[2] This assumption ensures that the robot does not become confused during navigation due to control errors. The value of ϵ describes the degree of inaccuracy in the robot's motions; thus, it is advantageous to make ϵ as large as possible without violating this assumption.

The existence of ϵ, together with its relationship with another constant, δ, allows the robot to detect proximity to a critical line using only the gap sensor. Although the robot lacks a map, it can still predict when certain critical events will happen; thus, the existence of ϵ is crucial to the efficacy of the on-line

[2]We define an equivalence relation among arrangements as follows. Let X, Y be two arrangements of walls, appear lines, and merge lines. If there is a bijection between the lines in X and the lines in Y such that the order of intersection of lines in X is preserved under the bijection, then X and Y are equivalent.

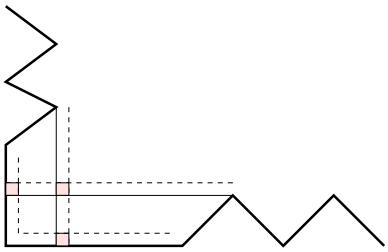


Figure 6: Several junctions. Thick solid lines are walls, thin solid lines are merge lines, and dashed lines are distance ϵ from merge lines or walls. A junction is an ϵ-by-ϵ parallelogram between intersecting dashed lines and intersecting merge lines. Each junction in this figure has been shaded, although not all junctions in the region are indicated.

algorithm.

2.4 Gap-Based Navigation

Suppose the pursuer is on the split slide of some merge line M, and is within ϵ distance of M. The pursuer will see two gaps, γ_1, and γ_2, corresponding to the regions hidden by the two points of tangency which define M. Let $\theta(\gamma)$ denote the angular position of a gap γ, relative to the direction of heading. We assume there exists some angle $\delta > 0$ such that $|\theta(\gamma_1) - \theta(\gamma_2)| < \delta$ if and only if the robot stays within ϵ distance of M. A discussion of the relationship between δ and ϵ appears in the appendix.

We call γ_1, γ_2 a *proximity pair* if the angular distance between them is less than δ, or if, as the robot moves, some gap γ splits into γ_1, γ_2. We define the direction of a proximity pair (relative to the robot's heading) as $\theta(\gamma_1, \gamma_2) = \frac{\theta(\gamma_1)+\theta(\gamma_2) \bmod 2\pi}{2}$. If the pursuer moves in directions $\theta(\gamma_1, \gamma_2) + \frac{\pi}{2}$ or $\theta(\gamma_1, \gamma_2) - \frac{\pi}{2}$, the gaps will either quickly diverge or quickly merge. On the other hand, if the pursuer moves in the directions $\theta(\gamma_1, \gamma_1)$ or $\theta(\gamma_1, \gamma_1) + \pi$, the gaps will remain within δ angular distance, but will not merge. This motion corresponds to moving along a merge line in R within ϵ distance. Small errors in control can be corrected by using feedback from the gap sensor to vary the commanded direction. For convenience, when the pursuer moves in direction $\theta(\gamma_1, \gamma_2)$, we say that it moves *towards the proximity pair* γ_1, γ_2. Similarly, when the pursuer moves in the direction $\theta(\gamma_1, \gamma_2) + \pi$, we say that it moves *against the proximity pair* γ_1, γ_2.

The pursuer's motion strategy will be described in terms of simple motion primitives. A primitive motion is movement either along a wall or along a merge line, executed by the robot in terms of movement towards or against a proximity pair. A primitive motion terminates when the pursuer reaches a *junction*, at which point another primitive motion command is issued.

As the pursuer moves along a wall, a junction is encountered when a proximity pair arises. If the pursuer is moving along a merge line, (that is, towards or against a proximity pair γ_1, γ_2), then a junction is encountered when the robot arrives at a wall, or when a new proximity pair (different from γ_1, γ_2) arises. Thus, a junction is an ϵ-by-ϵ parallelogram at the intersection of a merge line with another merge line or with a wall, always on the split side of a merge line. See Figure 6.

At each junction, one of four possible primitive motions is possible:

F Continue forward with the same heading.

B Move backwards with respect to the current heading

R Move to the right. If the pursuer arrived at a wall, then it turns right and follows the wall. Otherwise, the pursuer moves right either in the direction towards or against the new proximity pair (depending

on which one is to the right).

L Move to the left. This motion is symmetric to the R primitive.

From any initial position, assume the pursuer can be commanded to move until a junction is encountered. From this position and heading, a *motion strategy*, $\mu = \langle \mu_1, \mu_2, \ldots, \mu_m \rangle$ is defined as a finite sequence of primitive motions, in which each element $\mu_i \in \{F, B, R, L\}$. The algorithm presented in Section 3 will compute a strategy, μ, that will guarantee all evaders will be detected, if such a strategy exists. The strategy is computed online by making decisions based on the sensor history. The execution of the strategy depends only on the pursuer's ability to follow walls, track gaps and envision a solution based on its record of observations.

2.5 The Pursuit Status

Consider the circular sequence of gaps reported by the sensor at a junction j. Recall from Figure 2 that each gap corresponds to a hidden connected region of R that is not visible. For each hidden region, a binary label can be marked as 1 to indicate that the region is contaminated, and 0 to indicated it is cleared. Let B_j denote a circular sequence of binary labels, in which each label indicates the status of a gap detected by the gap sensor when the pursuer is at j. We say B_j is the *pursuit status*, and denote $B_j(\gamma)$ as the status of gap γ, at junction j.

Suppose that at least one gap event occurred when the robot moves from j to k, and B_j must be transformed to B_k to reflect the gap changes. If a gap γ disappears, then B_k excludes the label for γ. If a gap γ' appears, then $B_k(\gamma') = 0$ because the hidden region represented by γ was visible when the robot was at j, and therefore it contains no evaders. If gaps γ_1, γ_2 merge into γ', then $B_k(\gamma') = B_j(\gamma_1) \vee B_j(\gamma_2)$, in which $\vee$ denotes the logical OR operation. This notion is correct because one contaminated region could spread to other regions. If γ splits into γ'_1, γ'_2, then $B_k(\gamma'_1) = B_k(\gamma'_2) = B_j(\gamma)$.

3 The Algorithm

The algorithm consists of interleaved processes. In *exploration*, the robot travels through R to learn and record information about the environment, and in *envisioning*, the robot searches this information for a path which will clear R. It is possible to finish exploration first, and then to attempt to envision a solution; we interleave these processes for efficiency.

The navigation graph We define the *navigation graph*, g_n, as follows. Let g_n contain a vertex for each junction in the environment. If $u, v \in V(g_n)$, an edge connects them if the junction corresponding to v can be reached from the junction corresponding to u by a single primitive motion. Note that g_n is undirected, because the primitive motions are reversible. Since primitive motions correspond to following a wall or a merge line, it may be useful to think of uv as a straight line (or curved line, if the robot follows a wall to move from u to v) of ϵ width in R connecting the junctions corresponding to u and v.

Note that g_n is planar. Following an edge corresponds to moving along a wall or moving towards (or against) a proximity pair; therefore, the intersection of two edges corresponds to the point where, as the robot follows one of the edges, a new proximity pair is visible. This point is a junction, or a vertex in g_n. Two edges only intersect at a vertex; therefore, g_n is planar.

Because g_n is planar, it can be partitioned into faces. Consider a face, f, of g_n. We refer to the corresponding *face of* R as the area in R enclosed by the robot as it travels along the border[3] of f.

[3]We use the term "border" assuming its definition in planar graph theory; thus, an edge borders at most two distinct faces, and a face f borders another face f' if and only if f and f' share an edge.

At a given time, g_n represents the robot's knowledge about R; it might not fully represent the junctions of R (in fact, it usually does not). If the robot has fully explored R, then we refer to its navigation graph as G_n. Using this notation, g_n can be thought of as the portion of G_n that is revealed incrementally during exploration.

For convenience, we will discard the distinction between g_n and R and say that the robot is at a vertex in g_n when it is at the corresponding junction in R, the robot travels along the edge uv when it travels by a single primitive motion between the corresponding junctions, and the robot explores a face of g_n when it travels along the border of the corresponding face of R.

The status graph We define the *status graph*, g_s, which is constructed from the information in the navigation graph and the pursuit status. For each vertex in g_n and each possible pursuit status at that vertex, we construct a vertex in g_s; more formally, $V(g_s) = \{\langle v, B\rangle : v \in V(g_n), B \in \{0,1\}^{n_v}\}$, in which n_v is the number of gaps detected by the gap sensor when the pursuer is at v. If an edge connects two vertices u, v in g_n, and it is possible for a particular pursuit status B_u to be transformed into B_v after moving from u to v, then $(\langle u, B_u\rangle, \langle v, B_v\rangle) \in E(g_s)$. Note that g_s is a directed graph.

Although g_s is an exponential-size graph, we make some observations which suggest that graph search in g_s may be efficient. First, the edge set is usually sparse. In particular, let $\langle u, B\rangle$ be a vertex in g_s. There are at most four neighbors of u in g_n. One property of the pursuit status is that, given a starting vertex, u in g_n, a pursuit status B, and an edge uv in g_n, the pursuit status B' which results from traveling along uv is unique and is a deterministic function of the gap events which occur along uv. This property implies that $\langle u, B\rangle$ has exactly one neighbor among the vertices $\left\{\left\langle v, \{0,1\}^{|gaps(v)|}\right\rangle\right\}$, where v is some neighbor of u. Then the maximum degree of $\langle u, B\rangle$ is 4, so $|E(g_s)| = \frac{1}{2}\sum_{v \in V(g_s)} degree(v) \leq 2|V(g_s)|$.

Second, it is not necessary to store the entire graph at once; it can be lazily computed, given g_n, during graph search. This approach lead to good performance in [14], in which a similar exponential-size information state graph was searched to solve pursuit-evasion problems with a known map. In Section 5, we discuss an implementation of this algorithm which suggests that, despite the theoretical exponential size of the graph, the practical performance of the algorithm is efficient. In Section 6, we propose an extension of this work which would yield an on-line algorithm for a pursuer with two flashlights. It is possible that such an algorithm would yield a smaller status graph, because a pursuer with two flashlights has less information at any point in time than a pursuer with omnidirectional vision.

Let G_s denote the status graph which would be constructed from G_n.

Initialization To begin the exploration, the robot moves until it encounters a wall; then it moves left until it detects a proximity pair. Now it is at a junction; it initializes g_n with $V(g_n) = \{v\}$, $E(g_n) = \phi$, in which v is a vertex that represents the current junction.

At this point, at least two motions are possible: B (corresponding to motion that reverses the initialization) and L (corresponding to motion along the merge line which causes the proximity pair). The robot creates edges for the paths described by each possible primitive motion, and marks them *unexplored*. The edges are created in order of increasing clockwise angular distance from the direction of heading.

We now describe how the robot explores R, starting with the subprocedures of traveling along an edge and exploring a face.

Traveling along an edge. Let $gaps(u)$ be the circular sequence of gaps detected by the gap sensor at vertex u. As the robot travels along edge uv, it records or computes the following information:

1. The circular sequences $gaps(u)$, $gaps(v)$.

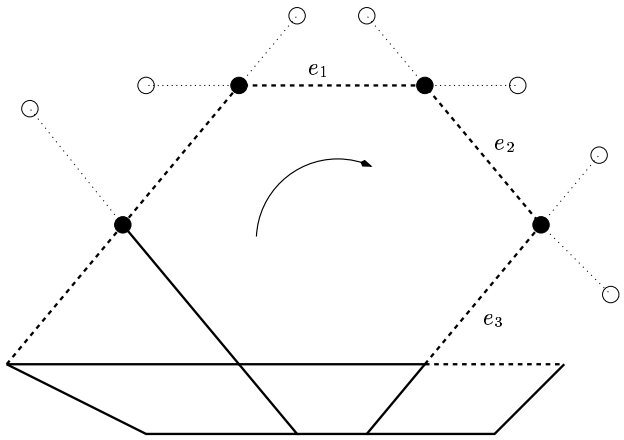


Figure 7: The result of a face exploration. Incomplete vertices are solid circles; unexplored edges are dotted lines; incomplete edges are thick dashed lines; complete edges are thick solid lines. The edges e_1, e_2, e_3 form an incomplete path. The arrow indicates the face which has been explored, and the direction of exploration. Note that only a part of g_n is shown.

2. The new gaps detected during the motion.

3. The gap events which occur, including the gaps involved in each event.

By our assumptions, the robot can easily compute or record items 1, 2, and 3. In Section 4.3, we mention other items the robot must record, which allow it to construct g_s.

Exploring a face Suppose that the pursuer is at a vertex v and is commanded to explore a face. It then executes the primitive motions $\langle L, L, \ldots, L \rangle$ until v is reached again. Once it returns to v, the robot has explored a face.

There are several ways to detect whether v has been reached again: 1) the robot can recognize this event by dropping a marker, or a "pebble," at v and detecting the marker upon return (as in [4]); 2) very coarse angular odometry could be used to stop after enough turns have been made; 3) the robot could exploit a distance bound by stopping after traveling for a certain time, if such a bound exists. It might be possible to determine that v has been reached again using only the gap sensor, but this problem remains open.

Suppose that the robot visits a vertex u during its exploration of a face. If the robot could execute a primitive motion at u without hitting a wall or exploring an edge it has previously explored, then it adds to g_n a new vertex, u', and an edge, uu', corresponding to this motion. It marks uu' as an *unexplored edge*. The vertex u is the *explored vertex* of the unexplored edge.

If a vertex has been explored, but is incident to an unexplored edge, then we say the vertex is *incomplete*. Similarly, if an edge has been explored, but it borders an unexplored face, then we say the edge is *incomplete*. Note that an incomplete vertex must be incident to an incomplete edge. An explored edge which borders two faces is *complete*.[4] Also, if a path consists entirely of incomplete edges, we say that it is an *incomplete path*. See Figure 7.

The global exploration strategy Face exploration proceeds in stages. In the first stage, a single face is explored, starting at the first junction found during initialization. For stage i, where $i > 1$, the robot visits

[4]Note that a single edge borders at most two faces, since g_n is planar.

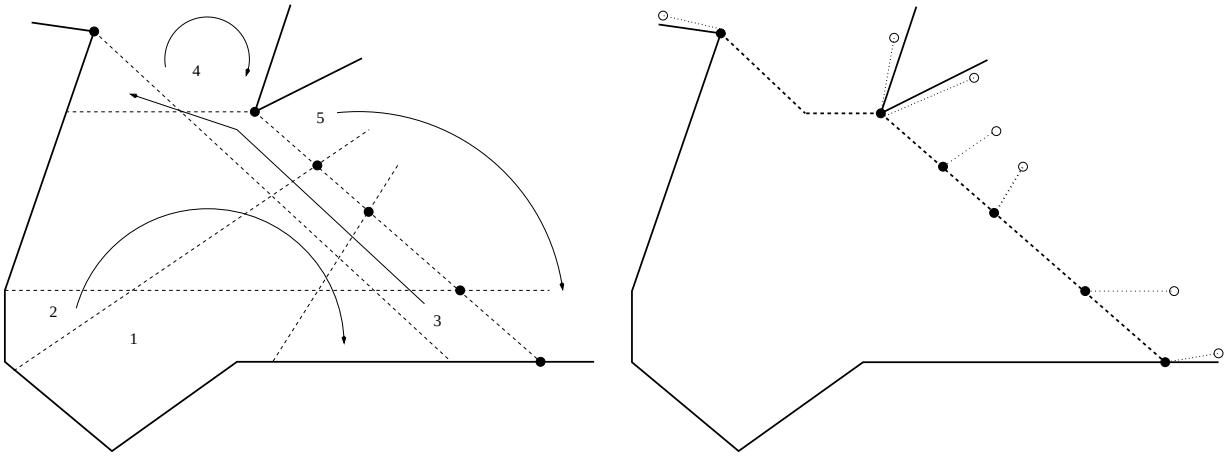


Figure 8: a) The global exploration strategy specifies that faces are explored in concentric waves, starting with the first face the robot explores. When a break in the environment is discovered (in stage 3), it is explored recursively (in stages 4 and 5). This strategy ensures that when the robot encounters a junction, it can determine whether or not it has previously visited the junction, and can thus build g_n accurately. Dashed lines indicate merge lines, and solid lines without arrows indicate parts of the wall. b) The state of the exploration is shown, following stage 3 of the global exploration algorithm. Thick dashed lines indicate incomplete edges, and solid circles indicate incomplete vertices. Dotted lines indicate unexplored edges, and hollow circles indicate unexplored vertices. Where unexplored edges correspond to parts of the wall, both are shown, but the unexplored edges are set apart from the wall for clarity. Note that only part of R is shown.

each incomplete vertex discovered in stage $i-1$; at each such vertex, the robot explores a face. In even stages, the incomplete vertices are visited in clockwise order; in odd stages, they are visited in counterclockwise order, to reduce the number of motions needed to begin the next stage. See Figure 8.

In some stages, there will be more than two incomplete vertices which are on ∂R. This situation indicates that a break in the environment has been discovered. In these situations, each part of the environment is explored recursively. Thus the explored region (that is, the union of the explored faces) is always simply connected.

Let the incomplete vertices in some iteration be $v_1, \ldots, v_m$. In odd stages (in which the robot visits the incomplete vertices in counterclockwise order), the robot explores the i^{th} face by traveling to v_i, dropping a pebble, and repeatedly issuing L motions until it returns to v_i. All of the edges and vertices explored during this process are added to g_n, except for the last l edges and $l+1$ vertices explored, where l is the length of the incomplete path from v_i to v_{i+1}.[5] In even stages, the robot explores the i^{th} face by traveling to v_i, dropping a pebble, taking the incomplete path from v_i to v_{i+1}, and issuing L motions until it returns to v_i. All of the edges and vertices encountered as the robot moves from v_{i+1} to v_i are added to g_n.

Envisioning and Terminating When the robot finishes exploring a face, it extends g_s to account for the new edges added to g_n, and the gap events which occur along those edges.[6] The robot then searches g_s

[5] In Section 4.2 we will show that there is a single incomplete path from v_i to v_{i+1}, and further that the edges on this path are the only explored edges on the border of f_i, where f_i is the face whose exploration begins at v_i.

[6] For example, suppose that when the robot explores f, it discovers the new vertices x and y, each having n gaps. Then it adds the set of vertices $\{(v, B) : v \in \{x, y\}, B \in \{0,1\}^n\}$ to $V(g_s)$. Also, suppose that the robot determines that if the robot starts at x, with pursuit status $\langle 0 \cdots 01 \rangle$, and travels to y, then its new pursuit status will be $\langle 0 \cdots 00 \rangle$. Then the robot will add the edge $(\langle x, \langle 0 \cdots 01 \rangle \rangle, \langle y, \langle 0 \cdots 00 \rangle \rangle)$ to $E(g_s)$.

for a path from its current junction and status to a junction and status of $\langle 0 \cdots 0 \rangle$. If such a path can be found, then it is appended to the previous history of motions as a path that clears R. If no such path can be found, and no unexplored edges remain, then the algorithm terminates, reporting that there is no path which can clear R.

4 Algorithm Analysis

In this section, we consider the theoretical properties of the algorithm. In Section 4.1, we show that if it is possible to clear R with a powerful robot (that is, one which has a map of R, perfect odometry, and a compass) then it is possible to clear R with the limited model we have chosen.

We then consider the exploration and envisioning stages of the algorithm separately. In Sections 4.2 and 4.3, we show that it is possible for the robot to navigate through R using g_n, and that it is possible for the robot to search g_s, thus finding a solution if it exists.

Section 4.1 is theoretically important, but may obfuscate the main ideas of the algorithm. It may be useful to read it briefly at first, and to read it more carefully later.

4.1 Feasibility

We now prove that it is possible to clear R using only the model described in Section 2.

Let $R' \subset R$ be the portion of the environment which is exactly ϵ distance from the wall, or exactly ϵ distance from a merge line (on the split side). More precisely, let the merge lines of R be $M_1, M_2, \ldots$. Recall that the part of R on the split side of M_i is denoted $R_{s(M_i)}$. Thus, $R' = \{x : d(x, \partial R) = \epsilon\} \cup \{x : \exists M_i \text{ such that } x \in R_{s(M_i)}, d(x, M_i) = \epsilon\}$, in which d is the Euclidean distance metric. We show that if it is possible to clear R, then it is possible to do so while moving only in R'.

Note that R' is a subset of the space in which the robot is allowed to move via primitive motions. However, this restriction of the robot's motion does not affect the information the robot collects. Let C be some critical line that the robot crosses while following a line K (which may be a wall or a merge line) at distance $\varepsilon < \epsilon$. By assumption the two arrangements described in Section 2.3 are equivalent; therefore the robot will cross C while following K at distance ϵ as well. On the other hand, if the robot crosses C while moving in R', then there exists a point $x \in C$ with $d(x, K) = \epsilon$. Since there are no walls or merge lines between x and K, then C also intersects K; therefore the robot will cross C if it follows K at a distance $\varepsilon < \epsilon$.

Lemma 4.1 *If π is a path which clears R, then there exists a path $\pi' : [t_s, t_f] \to R$ which clears R, such that $\pi'(t_s), \pi'(t_f) \in R'$.*

Proof: Let π' be the following path. The robot starts at some point in R'. From that point, it goes to $\pi(t_s)$, then executes π, and then goes to some other point in R'. The robot's motions before reaching $\pi(t_s)$ do not alter the fact that π clears R; once R has been cleared, it cannot be recontaminated. Therefore, π' clears R. ■

Now, suppose the robot executes a path $\pi : [t_0, t_3] \to R'$ which can be described as follows. For some merge line M, we have $\pi(t_0), \pi(t_3) \in R - R_{s(M)}$; furthermore, there exist t_1, t_2 such that $t_0 < t_1 \leq t_2 < t_3$, and $\pi : [t_1, t_2] \to R' \cap R_{s(M)}$, but at all other times $t \notin [t_1, t_2]$, $\pi(t) \in R - R_{s(M)}$. An example of such a path appears in Figure 9.

We call such a path a *splitting hop around M*. Since the robot moves continuously, then the robot crosses M (from the merged side to the split side) at t_1, and then crosses M again at t_2 (from the split side to the merged side). Let α be the gap which splits at t_1, and β be the gap which merges at t_2.

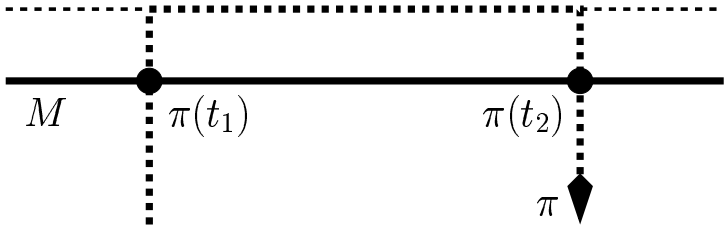


Figure 9: An example of a hop around M. Dashed lines are in R'. Solid circles indicate $\pi(t_1)$, $\pi(t_2)$.

Lemma 4.2 *No recontamination occurs due to a hop around M; that is, $B_{\pi(t_2)}(\beta) \leq B_{\pi(t_1)}(\alpha)$.*

Proof: Let $split(\alpha) = (\gamma_1, \gamma_2)$. First, notice that neither γ_1 nor γ_2 can participate in any merge or split events between t_1 and t_2. Because α splits when the robot crosses M, and the robot stays within ϵ distance of M on the interval $[t_1, t_2]$, then γ_1 and γ_2 remain within δ angular distance of each other on the gap sensor. If γ_1 and γ_2 merge with each other, then the robot must have crossed M again; that is, γ_1 and γ_2 can only merge to form β.

Suppose, without loss of generality, γ_1 merges with some gap (other than γ_2) on the interval (t_1, t_2); call this gap γ_3. Before γ_1 and γ_3 merge, they must appear within δ angular distance on the gap sensor. Then γ_1, γ_3 form a proximity pair; but at the same time, γ_1, γ_2 are in a proximity pair, so there are three gaps in a single proximity pair, indicating a tri-tangent. This conclusion violates our general position assumption; therefore, γ_1 does not participate in any merge events, unless it merges with γ_2.

Similarly, suppose $split(\gamma_1) = (\gamma_3, \gamma_4)$ on the interval $[t_1, t_2]$. Just before the split, γ_1 and γ_2 are within δ angular distance on the gap sensor. Then after the split, both γ_3, γ_4 are within δ distance of γ_2 on the gap sensor; again, three gaps appear in a single proximity pair, violating our general position assumption.

Since obviously neither γ_1 nor γ_2 can participate in appearance events on the interval $[t_1, t_2]$, the only gap events γ_1 and γ_2 can participate in on this interval are disappearances.

Suppose that neither γ_1 nor γ_2 disappears on the interval $[t_1, t_2]$. Then $B_{\pi(t_1)}(\gamma_i) = B_{\pi(t_2)}(\gamma_i)$, for $i \in \{1, 2\}$. Thus we have:

$$B_{\pi(t_2)}(\beta) = B_{\pi(t_2)}(\gamma_1) \vee B_{\pi(t_2)}(\gamma_2) = B_{\pi(t_1)}(\gamma_1) \vee B_{\pi(t_1)}(\gamma_2) = B_{\pi(t_1)}(\alpha).$$

On the other hand, suppose that exactly one of γ_1, γ_2 disappears; assume that it is γ_1 without loss of generality. Let γ_3 be the gap which merges with γ_2 to form β. If γ_3 was visible at t_1, then γ_3 would have also split from α (since γ_3, γ_2 merge when the robot crosses M). Since γ_3 did not split from α, γ_3 must appear on the interval $[t_1, t_2]$. Since γ_3 appears, we have $B_{\pi(t_2)}(\gamma_3) = 0$; thus,

$$B_{\pi(t_2)}(\beta) = B_{\pi(t_2)}(\gamma_3) \vee B_{\pi(t_2)}(\gamma_2) = 0 \vee B_{\pi(t_2)}(\gamma_2) =$$

$$B_{\pi(t_2)}(\gamma_2) = B_{\pi(t_1)}(\gamma_2) = B_{\pi(t_1)}(\alpha).$$

Finally, if both γ_1, γ_2 disappear, then $\beta = merge(\gamma_3, \gamma_4)$, for some gaps γ_3, γ_4 which both appear on the interval $[t_1, t_2]$. Since $B_{\pi(t_1)}(\alpha) \in \{0, 1\}$,

$$B_{\pi(t_2)}(\beta) = B_{\pi(t_2)}(\gamma_3) \vee B_{\pi(t_2)}(\gamma_4) = 0 \vee 0 = 0 \leq B_{\pi(t_1)}(\alpha).$$

■

Consider also a *merging hop around M*, the dual of a splitting hop around M. That is, a merging hop is a path on R', starting at $R_{s(M)}$, proceeding to $R - R_{s(M)}$ at t_1, and returning again to $R_{s(M)}$ at t_2. It is

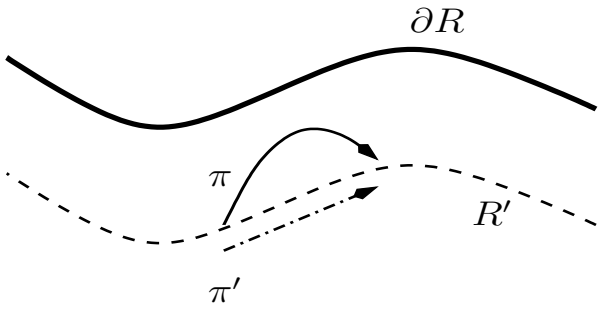


Figure 10: Lemma 4.3, Case 1. The image of π goes between R' and ∂R. The image of π' has been set apart from R' for clarity.

clear that the only recontamination which may occur on such a path is any recontamination which occurs at $\pi(t_1)$.

For any pair of statuses $B_j = \langle b_1 \ \cdots \ b_m \rangle$, $B_k = \langle b'_1 \ \cdots \ b'_m \rangle$, we say that B_k *dominates* B_j if, for all i, $(b_i = 0) \Rightarrow (b'_i = 0)$.

Lemma 4.3 *For any path $\pi : [t_s, t_f] \rightarrow R$, such that $\pi(t) \in R'$ if and only if $t \in \{t_s, t_f\}$, there exists a path $\pi' : [t_s, t'_f] \rightarrow R'$ such that $\pi'(t_s) = \pi(t_s), \pi'(t'_f) = \pi(t_f)$, and $B_{\pi'(t'_f)}$ dominates $B_{\pi(t_f)}$.*

Proof: By definition, $B_{\pi'(t'_f)}$ dominates $B_{\pi(t_f)}$ if and only if π' clears all the gaps that π clears, without contaminating any gaps that π did not contaminate. We consider the two kinds of paths π.

Case 1. The path π goes between R' and K, in which K is either a merge line or a part of ∂R. See Figure 10. In this case, π follows K at a distance $\varepsilon < \epsilon$. Let π' be the path which follows K at distance exactly ϵ. While executing π', the robot will cross the same appear and merge lines as it would while executing π. So $B_{\pi'(t'_f)} = B_{\pi(t_f)}$; then $B_{\pi'(t'_f)}$ dominates $B_{\pi(t_f)}$.

Case 2. There exists a minimal[7] simple closed loop, $L' \subset R'$, such that $\pi(t_s), \pi(t_f) \in L'$, and $\pi(t)$ is in the region of R bounded by L', for $t_s < t < t_f$. See Figure 11. Note that L' is a finite collection of lines in R'. Let these lines be enumerated in clockwise order as $L'_1, \ldots, L'_n$, starting with the line containing $\pi(t_s)$. Let L'_r be the line which contains $\pi(t_f)$. Let π' be the path which starts at $\pi(t_s)$, travels in the clockwise direction along $L'_1, L'_2, \ldots, L'_r$, and stops at $\pi(t_f)$.

Let R_{loop} be the region of R bounded by L'. For each L'_i, let L_i be the line in R which is distance ϵ from L'_i (that is, L_i is either a merge line or a wall). If π crosses an appear line A inside R_{loop}, then A also intersects the boundary of R_{loop} (that is, L') at two points, one of which is visited on a clockwise path between $\pi(t_s)$ and $\pi(t_f)$. Since π' is just such a clockwise path, π' also crosses A, thereby clearing any gaps which π clears.

Now we show that no recontamination occurs on π' that does not occur on π. Consider some L_i on the "interior" of R_{loop} (that is, some L_i such that $\exists x : x \in R_{loop} \cap L_i$). When the robot travels along L'_{i-1}, L'_i, L'_{i+1} during execution of π, it is executing a splitting hop around L_i. By Lemma 4.2, the gap which splits as a result of crossing L_i is not recontaminated.

Any recontamination which occurs is due to π' crossing some merge line $M \notin \{L_1, \ldots, L_n\}$. Suppose π does not cross M. The line M can be used to define a closed loop whose enclosed region contains π, but is smaller than R_{loop}, so L' is not minimal. This contradiction implies that π crosses M, thereby recontaminating the same gap which π' recontaminates when crossing M. Thus, π' clears any gaps which π clears, and does not recontaminate any gaps, except for those which π recontaminates, as desired.

[7] That is, there is no loop which encloses a smaller region of R and still meets the conditions stated.

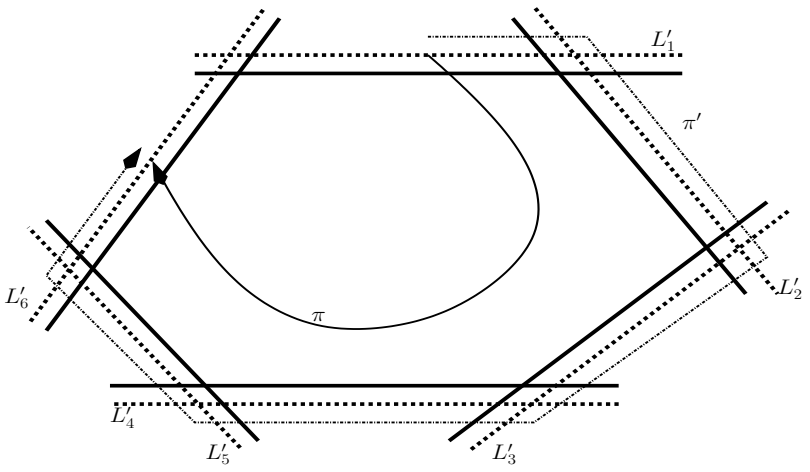


Figure 11: Lemma 4.3, Case 2. Merge lines are solid, and lines in R' (that is, lines ϵ distance from a merge line) are dashed. The path π' is set apart from R' for clarity.

Combining Lemmas 4.1 and 4.3, we have the following result: if there is some path that clears R, then there exists some path whose image lies in R' that also clears R.

Theorem 4.4 *If $\exists \pi : [t_s, t_f] \to R$ such that $B_{\pi(t_f)} = \langle 0 \cdots 0 \rangle$, then $\exists \pi' : [t'_s, t'_f] \to R'$ such that $B_{\pi'(t'_f)} = \langle 0 \cdots 0 \rangle$*

Proof: By Lemma 4.1, there exists a path $\tilde{\pi} : [\tilde{t}_s, \tilde{t}_f] \to R$ such that $\tilde{\pi}(\tilde{t}_s), \tilde{\pi}(\tilde{t}_f) \in R'$ and $\tilde{\pi}$ clears R. Subdivide $\tilde{\pi}$ into paths $\tilde{\pi}_0 : [\tilde{t}_s, \tilde{t}_1) \to R', \tilde{\pi}_1 : [\tilde{t}_1, \tilde{t}_2) \to R, \ldots, \tilde{\pi}_m : [\tilde{t}_m, \tilde{t}_f] \to R'$, so that the paths alternately map onto R' and R. Consider some $\tilde{\pi}_i : [\tilde{t}_i, \tilde{t}_{i+1}) \to R$. By Lemma 4.3, there exists another path $\pi'_i : [t'_i, t'_{i+1}) \to R'$ such that $B_{x'_i}$ dominates $B_{\tilde{x}_i}$ (in which $x'_i = \pi'_i(t'_{i+1})$, $\tilde{x}_i = \tilde{\pi}_i(\tilde{t}_{i+1})$). For all i, let $\pi'_i = \pi_i$ if $\pi_i \to R'$, and let π'_i be the path described above otherwise. Concatenate the π'_i into $\pi' : [t'_s, t'_f] \to R'$. By Lemma 4.3, $B_{\pi'(t'_f)}$ dominates $B_{\tilde{\pi}(\tilde{t}_f)}$. Since $\tilde{\pi}$ clears R, $B_{\pi'(t'_f)} = \langle 0\, 0 \cdots 0 \rangle$; therefore, π' clears R. ■

4.2 Exploration

Our algorithm relies on the correct computation of g_n and g_s; if these two graphs are correctly computed, then the algorithm itself is correct. We show that the robot can compute these graphs correctly using only the gap sensor.

First, we turn to the computation of g_n. The computed graph is correct if and only if there is a bijection between the explored junctions in R and the explored vertices in g_n such that junction adjacencies via motion primitives are preserved under the bijection as edges in g_n.

Since the robot always creates vertices in g_n when it visits new junctions in R, and similarly creates edges in g_n, the bijection condition is satisfied if and only if the robot neither neglects to visit some junction in R nor creates more than one vertex (or edge) for each junction (or single motion primitive between junctions) in R. The robot does not neglect junctions in R due to the nature of the global exploration strategy: every unexplored edge which can be seen in the k^{th} iteration of the strategy is explored in the $(k+1)^{th}$ iteration.

We now show that the robot does not create excessive vertices or edges. New vertices and edges are created only during face exploration; except for a single chain of edges between successive incomplete vertices, all edges and vertices visited are added to g_n.

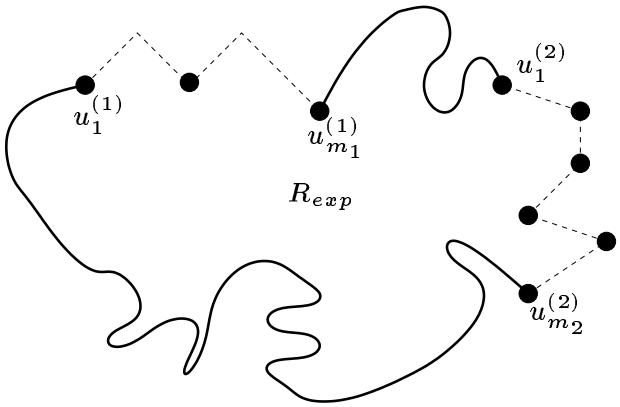


Figure 12: The explored region of R is bordered by incomplete paths and the explored parts of ∂R. Dashed lines indicate incomplete edges, solid lines indicate parts of ∂R, and solid circles indicate incomplete vertices. The paths from $u_1^{(1)}$ to $u_{m_1}^{(1)}$ and from $u_1^{(2)}$ to $u_{m_2}^{(2)}$ are incomplete paths. Since there are more than two vertices at a wall – $u_1^{(1)}, u_{m_1}^{(1)}, u_1^{(2)}$, and $u_{m_2}^{(2)}$ – there is a break in the environment. Each incomplete path borders a region which will be explored recursively.

The next lemma shows that in any unexplored face f, there is a single chain of explored edges on the border of f; these are the only edges visited during exploration of f which should not be added to g_n. Lemma 4.8 shows that this chain of explored edges is the same as the set of edges which the robot neglects to add to g_n after exploring f, implying that excessive edges are not added to g_n.[8]

Lemma 4.5 *Let $e_1, \ldots e_m$ be the edges on the border of an unexplored face f, enumerated in counterclockwise order, beginning with an unexplored edge incident to an explored edge. There exists an integer i such that $e_1, \ldots, e_i$ are unexplored, and $e_{i+1}, \ldots, e_m$ are explored.*

Proof: Consider the region of explored faces of R; denote this region as R_{exp}. The global exploration strategy ensures that R_{exp} is simply connected. Therefore, the edges on its border must either correspond to parts of ∂R, or must be incomplete edges. Furthermore, it is impossible for an incomplete edge to lie on the interior of R_{exp}, because if it did, then it would border two explored edges. Thus, the border of R_{exp}, which we refer to as the "frontier of exploration," is composed of the explored parts of ∂R, together with chains of incomplete edges. See Figure 12.

Now, suppose there is some face f for which the claim does not hold. The border of f is composed of at least two chains of unexplored edges and at least two chains of explored edges. That is, there exist i_1, i_2, i_3 such that $1 < i_1 < i_2 < i_3 < m$ and $e_1, \ldots, e_{i_1}$ are unexplored, $e_{i_1+1}, \ldots, e_{i_2}$ are explored, $e_{i_2+1}, \ldots, e_{i_3}$ are unexplored, and $e_{i_3+1}, \ldots, e_m$ are explored. See Figure 13.

Since $e_{i_1+1}, \ldots, e_{i_2}$ and $e_{i_3+1}, \ldots, e_m$ border an unexplored face, they must be incomplete. Incomplete edges cannot lie on the interior of R_{exp}; therefore these edges lie on the border of R_{exp}. In particular, there must be a path p of edges on the border of R_{exp} which includes both e_{i_2} and e_{i_3+1}.

Suppose that p is incomplete. Let q be the path $e_{i_2+1}, \ldots, e_{i_3}$, and let $C = p \cup q$. The set of edges C is a simple closed curve consisting of finitely many segments; therefore, it partitions the plane into two regions. Let R_C be the region inside of C. The region R_C is unexplored because the path q, which borders R_C, is composed of unexplored edges. See Figure 14(a).

Consider the state of exploration just after f has been explored. The new explored region is $R'_{exp} = R_{exp} \cup f$. The region R_C remains unexplored. Since p borders R_{exp} and q borders f, $C = p \cup q$ borders R'_{exp}.

[8] Although we only discuss edges here, we could similarly conclude that excessive vertices are not added to g_n.

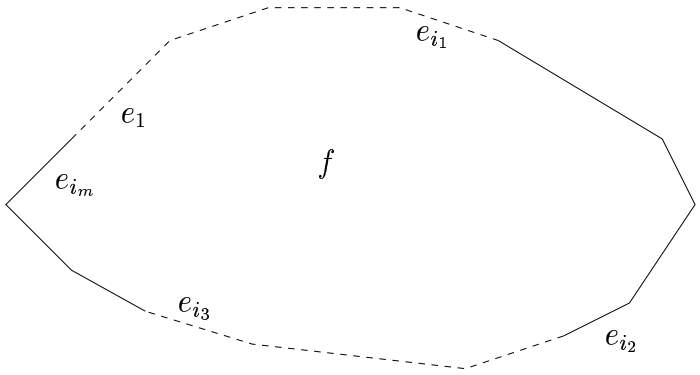


Figure 13: The face f has more than one chain of explored edges: $e_1, \ldots, e_{i_1}$ are unexplored, $e_{i_1+1}, \ldots, e_{i_2}$ are explored, $e_{i_2+1}, \ldots, e_{i_3}$ are unexplored, and $e_{i_3+1}, \ldots, e_{i_m}$ are explored. Dashed lines indicate unexplored edges, and solid lines indicate explored edges.

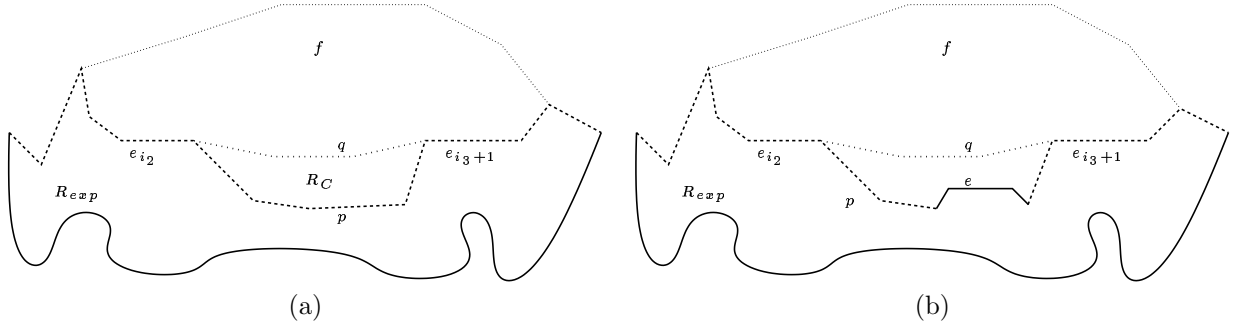


Figure 14: a) The path p is an incomplete path that includes e_{i_2} and e_{i_3+1}. The path q is the set of unexplored edges $e_{i_2+1}, \ldots, e_{i_3}$. Together they enclose an unexplored region R_C. After f is explored, R_C is a hole in the explored region $R_{exp} \cup f$. b) The path p of explored edges including e_{i_2} and e_{i_3+1} must include an edge e which corresponds to a part of ∂R. Then there is a part of ∂R on the interior of R, so R is multiply-connected. In both a) and b), dotted lines indicate unexplored edges, dashed lines indicate incomplete edges, and solid lines indicate explored parts of ∂R.

The region R_C is inside of C; therefore, R_C is an unexplored region on the interior of R'_{exp}. This implies that R'_{exp} is multiply-connected. However, the global exploration strategy ensures that R'_{exp} is simply connected. Thus, p must not be incomplete; p includes at least one complete edge.

Recall that all of the complete edges on the border of R_{exp} correspond to parts of ∂R. Hence there is some edge $e \in p$ which corresponds to a part of ∂R. Also, p lies on the interior of $R_{exp} \cup f \subseteq R$. Therefore, e lies on the interior of R. See Figure 14(b). This implies that R is multiply-connected, because e corresponds to a part of ∂R. However, R is simply-connected by assumption. Therefore, f must not have more than one chain of explored edges. ■

Now consider the state of the exploration just after the i^{th} stage of the algorithm. Let the paths of incomplete edges on the border of R_{exp} be $p_1 = u_1^{(1)}u_2^{(1)}, \ldots, u_{m_1-1}^{(1)}u_{m_1}^{(1)}$, $p_2 = u_1^{(2)}u_2^{(2)}, \ldots, u_{m_2-1}^{(2)}u_{m_2}^{(2)}$, ... Since the vertices $u_1^{(j)}$, $u_{m_j}^{(j)}$ are incident both to edges that correspond to ∂R and to incomplete edges, they are incomplete vertices. Moreover, if there is more than one incomplete path on the border of R_{exp}, then there is a break in the environment, and the unexplored region bordered by each incomplete path will be explored recursively, as specified by the global exploration strategy. See Figure 12.

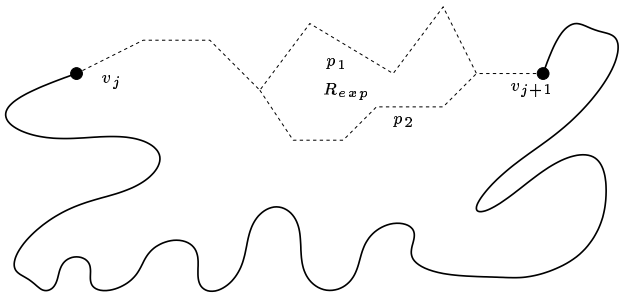


Figure 15: There cannot be two incomplete paths p_1 and p_2 from v_j to v_{j+1}, since one path must have an edge which lies on the interior of R_{exp}, which must be a complete edge. In this case, since the edges on p_1 border an explored region, p_2 must lie inside R_{exp}. Dashed lines indicate unexplored edges, and solid lines indicate ∂R.

Since each incomplete vertex must be adjacent to an incomplete edge, all of the incomplete vertices are on the frontier of exploration. We combine this fact with the foregoing discussion to achieve the following goal. For each face f explored in the $(i+1)^{th}$ stage of the algorithm, we can determine which previously-explored edges form f's chain of unexplored edges.

The following observation is immediate from the definition of explored edges, incomplete edges and incomplete vertices, and Lemma 4.5.

Observation 4.6 *The chain of explored edges of an unexplored face is an incomplete path, and its endpoints are incomplete vertices.*

We will claim that the chain of explored edges of an unexplored path is an incomplete path between consecutive incomplete vertices. We now prove that there is exactly one such incomplete path.

Lemma 4.7 *Let the incomplete vertices explored in the $(i+1)^{th}$ stage be $v_1, \ldots, v_m$. There is exactly one incomplete path between v_i and v_j, for $1 \leq i, j \leq m$.*

Proof: Since each incomplete vertex is incident to an unexplored edge, then each incomplete vertex is on the frontier of exploration. Since the frontier of exploration is composed of explored parts of ∂R and incomplete paths, each incomplete vertex must lie on one of these incomplete paths.

Note that if there are more than two incomplete paths on the border of R_{exp}, then each incomplete path borders a region which will be explored recursively. See Figure 12. This implies that if there are two incomplete vertices which are explored in the same stage of global exploration, then they must lie on the same incomplete path on the border of R_{exp}. Hence, for $1 \leq i, j \leq m$, v_i and v_j lie on the same incomplete path on the border of R_{exp}; it follows that there is at least one incomplete path between v_i and v_j.

If there were more than one incomplete path from v_i to v_j, then one path would be on the interior of R_{exp}, and would thus contain complete edges. See Figure 15. Thus, there is no more than one incomplete path from v_i to v_j.

Therefore, there is exactly one incomplete path from v_i to v_j, for $1 \leq i, j \leq m$. ■

Let f_j be the j^{th} face explored in this stage.

Lemma 4.8 *The chain of explored edges of f_j is the incomplete path from v_j to v_{j+1}.*

Proof: First, we show that the incomplete path from v_j to $v_{j'}$, where $|j - j'| > 1$, cannot be the border of any face. Suppose that such a path were the border of a face f, and assume that $j < j'$, without loss of

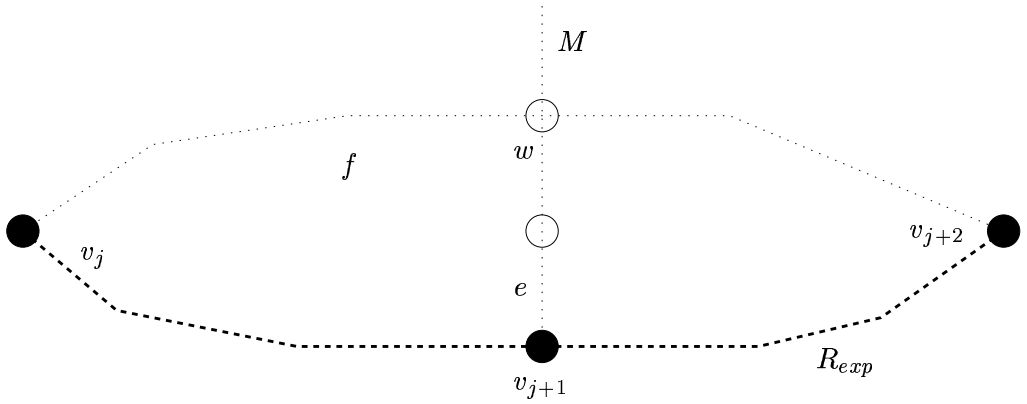


Figure 16: A face whose chain of explored edges is the incomplete path from v_j to v_{j+2}. The unexplored edge incident to v_{j+1} must be on the interior of f; this fact implies that a merge line M pierces f in two places, v_{j+1} and w, which implies that f is not a face. Thick dashed lines indicate incomplete edges, dotted lines indicate unexplored edges, solid circles indicate incomplete vertices and hollow circles indicate unexplored vertices.

generality. Consider the incomplete vertex v_{j+1}, and the unexplored edge e incident to it. Since v_{j+1} is on the incomplete path from v_j to $v_{j'}$, the incomplete edges incident to v_{j+1} border f. See Figure 16.

But the edge e must be inside one region bordered by the incomplete path from v_j to $v_{j'}$. This path borders two regions: R_{exp} and f. The edge e is unexplored, so e is not on the interior of R_{exp}. Hence e is on the interior of f. Note that since e is an edge of g_n, and e does not correspond to motion along ∂R, that it must correspond to a primitive motion along some merge line, M, of R. Since merge lines are bounded only by ∂R, this merge line must pierce the boundary of f in two places: at v_{j+1} and at some other vertex w. Then there is a path from v_{j+1} to w which lies on the interior of f. Therefore f is not a face, or g_n is not planar, both of which are contradictions.

The remainder of the proof is by induction on j. For $j = 1$, the chain of explored edges for f_1 must be the incomplete path from v_1 to v_2, since there is no other vertex $v_{j'}$ such that $|1 - j'| = 1$. For $j > 1$, the exploration of f_j begins at v_j, as specified by the algorithm. The chain of explored edges of f_j must be either the incomplete path from v_{j-1} to v_j or the incomplete path from v_j to v_{j+1}, since there are no other vertices $v_{j'}$ such that $j - j'| = 1$. Since the incomplete path from v_{j-1} to v_j is the chain of explored edges of f_{j-1} by the inductive hypothesis, the explored edges of f_j must be the incomplete path from v_j to v_{j+1}. ■

Theorem 4.9 *There is a bijection from the junctions in R to the vertices in g_n, such that junction adjacencies via single motion primitives are preserved under the bijection as edges in g_n; that is, g_n is correctly constructed.*

Proof: As discussed at the beginning of this section, g_n will be correctly constructed if and only if the robot does not create excessive vertices or edges in g_n after exploring a face f. According to Lemma 4.5, there is precisely one chain of explored edges on the border of any unexplored face; the robot must add all edges and vertices visited during exploration of f to g_n, except for vertices and edges on this chain. According to Lemma 4.8, this chain of explored edges is correctly computed, and is identical to the set of edges which the robot does not add to g_n, as specified by the exploration algorithm. Therefore, no excessive vertices or edges are created in g_n. Hence, g_n is correctly constructed. ■

4.3 Envisioning

Suppose the robot is at vertex v_0, with pursuit status B_{v_0}. We wish to determine – without issuing any motion commands – what the new pursuit status would be if the robot executed some walk in g_n. We call this the *future status* problem.

Suppose the robot has an ideal compass, and can perfectly measure its direction of heading with respect to absolute North; then it can solve this problem easily. Let $gaps_\theta(v)$ be the gap sensor report at vertex v, when the robot is oriented in the direction θ. The angle θ is measured relative to absolute North, which is 0 radians, and the gaps are enumerated in the clockwise direction starting at θ.

Let the walk in g_n be denoted $v_0v_1, \ldots, v_{m-1}v_m$. Assume that the robot is at v_0, oriented in the direction θ. Let θ_0 be the direction of heading when the robot first entered v_0 along the edge v_0v_1.

By re-enumerating the gaps in $gaps_\theta(v)$, the robot can compute $gaps_{\theta_0}(v_0)$. If the gap events that occur along v_0v_1 are recorded according to the enumeration of the gaps in $gaps_{\theta_0}(v_0)$, then the robot can compute the status B_{v_1} that would result from moving along v_0v_1. The process repeats recursively until the robot has computed B_{x_m}.

We replace the ideal compass using two main ideas. First, the robot uses a "local compass," which we call a *stable gap*, to provide a fixed frame of reference with respect to motion along a particular edge. Second, a *gap sensor distance* measure replaces unreliable angular measurements with reliable gap counting.

4.3.1 Stable Gaps

Suppose the robot moves from junction u to junction v. Consider some $\gamma \in gaps(v)$. We say that γ is *stable with respect to uv*, which we denote $stable_{uv}(\gamma) = true$, if one of the following conditions hold while the robot moves along uv:

1. $\neg appears_{uv}(\gamma)$
2. $\gamma = merge_{uv}(\alpha, \beta)$ for some α, β
3. $split_{uv}(\alpha) = (\beta, \gamma)$ for some α, β

Above, we extend our earlier notation, so that, for example, $appears_{uv}(\gamma)$ means "as the robot moves from u to v, gap γ appears." If we identify α, β and γ whenever $split(\gamma) = (\alpha, \beta)$ or $\gamma = merge(\alpha, \beta)$, then we have $stable_{uv}(\gamma) = stable_{vu}(\gamma)$ for all edges uv and all gaps γ.

Lemma 4.10 *For every pair of junctions (u, v), $\exists \gamma \in gaps(v)$ such that $stable_{uv}(\gamma) = true$, or the robot can clear R by traveling along uv.*

Proof: The robot crosses at most 2 merge lines when traveling along uv: at most one just after leaving u, and at most one just before entering v. We consider the claim under all possible cases. See Figure 17.

Case 1. The robot crosses no merge lines when traveling along uv; therefore, no gaps merge or split during motion along uv; the only gap events are appearances and disappearances. Suppose there does not exist a gap γ as described. Then for all $\gamma \in gaps(v)$, γ appears as the robot moves along uv. Appearing gaps receive a "0" status label, and disappearing gaps have no status label; therefore, after the robot moves along uv, $B_v = \langle 0 \cdots 0 \rangle$, and R is cleared.

Case 2. The robot crosses a merge line just after leaving u. Let $\gamma = merge(\alpha, \beta)$ for some $\alpha, \beta \in gaps(u)$. Since merged gaps never disappear, $\gamma \in gaps(v)$, which implies that $stable_{uv}(\gamma) = true$.

Case 3. The robot crosses a merge line just before entering v. Let $split(\alpha) = (\beta, \gamma)$ for some α, β. The gap γ is in the gap sensor just after the robot crosses the merge line, and the robot crosses the merge line just before entering v; therefore $\gamma \in gaps(v)$, which implies that $stable_{uv}(\gamma) = true$.

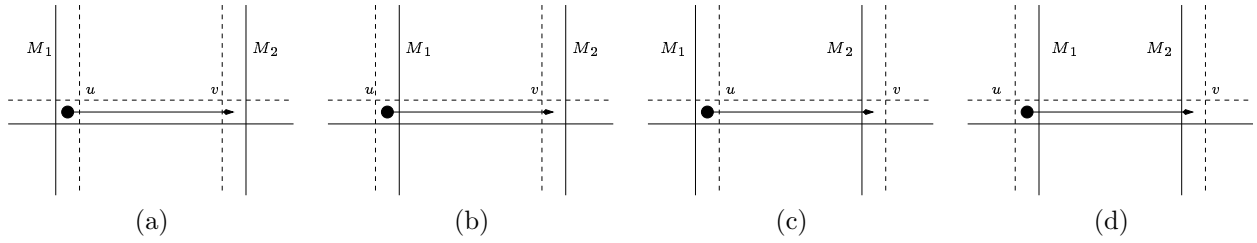


Figure 17: Lemma 4.10: a) Case 1. The robot crosses no merge lines when going from u to v. b) Case 2. The robot crosses M_1 just after leaving u. c) Case 3. The robot crosses M_2 just before entering v. d) Case 4. The robot crosses M_1 just after exiting u, and crosses M_2 just before entering v.

Case 4. The robot crosses a merge line just after leaving u and just before entering v. Both the events in Cases 2 and 3 occur, and there is at least one stable gap. ■

For all edges e, let σ_e be the gap such that $stable_e(\sigma_e) = true$. If there are multiple such gaps for a single edge, the robot chooses one, with preference to merged gaps.

After traveling along uv for the first time, the robot computes and records σ_{uv} as follows. If a merge line is crossed, then the robot can determine α, β, γ such that either $split_{uv}(\alpha) = (\beta, \gamma)$ or $\gamma = merge_{uv}(\alpha, \beta)$; γ is recorded as a stable gap. If a merge line is not crossed, the robot checks whether $\exists \gamma \in gaps(u) \cap gaps(v)$. If so, then γ neither appears nor disappears as the robot moves along uv, which implies that γ is a stable gap. If there is no such gap γ, then there is no stable gap, and the robot has cleared R.

4.3.2 Gap Sensor Distance

The robot can record gap events which occur along an edge uv by enumerating $gaps(u)$ and $gaps(v)$ with σ_{uv} first. This record can be used to solve the future status problem, assuming that the robot can identify, when it is at u or v, which gap is σ_{uv}.

Suppose the robot travels along uv, and determines which gap is the stable gap σ_{uv}. Now suppose that, at some point in the future, the robot returns to v. We wish to identify which of the gaps in the gap sensor is σ_{uv}; we call this problem the *future identification* problem.

The difficulty, again, is that the robot lacks a compass and perfect odometry. Suppose that when the robot first determines which gap is σ_{uv}, it also observes σ_{uv} appears at $\frac{\pi}{4}$ radians in the clockwise direction from absolute North. Then identifying σ_{uv} on future trips to v is trivial: the robot orients itself towards absolute North, and finds the gap which appears at $\frac{\pi}{4}$ radians in the clockwise direction.

We use the proximity pairs to solve this problem. Recall that the robot can always identify proximity pairs, and further that at each junction, either one or two proximity pairs are visible. For gaps α, β, we define the *gap sensor distance* $\tau_v(\alpha, \beta)$ as the number of gaps between α and β in the gap sensor report at v, in the clockwise direction. If $gaps(v) = \gamma_1, \ldots, \gamma_m$, then $\tau_v(\gamma_i, \gamma_j) = j - i \bmod m$.

Suppose that there is only one proximity pair at a junction v, and its rightmost gap is γ. When the robot travels along an edge uv and determines which gap is σ_{uv}, it computes $x = \tau_v(\gamma, \sigma_{uv})$. On a future visit to v, the robot can find γ (because it can detect the single proximity pair at v) and can find the gap whose gap sensor distance from γ is x. This gap must be σ_{uv}. See Figure 18.

Suppose that there are two proximity pairs at v, with rightmost gaps α and β. Then v is at the intersection of two merge lines, and when the robot arrives at v, it must be following one of these merge lines, which means that it must be moving towards or away from one of the two proximity pairs. If the robot travels along the edge uv to arrive at v, and this motion corresponds to moving towards (or away from) a proximity

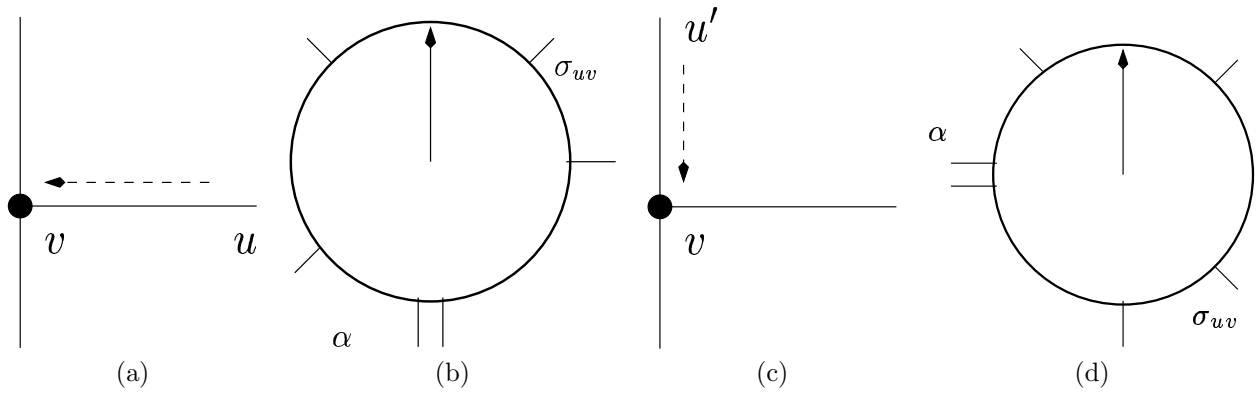


Figure 18: Solving the future identification problem for a vertex with one proximity pair, α. a) The first visit to v via the edge uv. b) The robot's gap sensor after the motion in a). The arrow indicates the direction of heading. The robot computes $\tau_v(\alpha, \sigma_{uv}) = 3$. c) A future visit to v, via another edge $u'v$. d) The robot's gap sensor after the motion in c). The robot identifies σ_{uv} as the gap which is 3 gaps in the clockwise direction from α on the gap sensor.

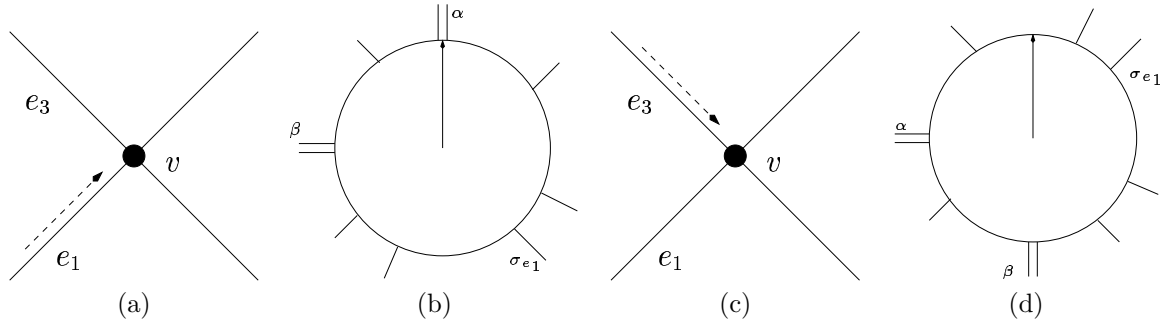


Figure 19: Solving the future identification problem for two proximity pairs. a) The first visit to v via e_1. b) The robot's gap sensor after the motion in a). The arrow points in the direction of heading. The robot records $\tau_v(\alpha, \sigma_{e_1}) = 3$ and $\tau_v(\beta, \sigma_{e_1}) = 5$. c) A future visit to v via e_3. Note that β controls this motion. d) The robot's gap sensor after the motion in c). The robot identifies σ_{e_1} as the gap which is 5 gaps in the clockwise direction from β.

pair whose rightmost gap is α, we say that α controls the motion on uv.

Suppose that the robot travels along edge e_1 to arrive at v and α controls the motion along e_1. Let e_2 be the edge which corresponds to the F motion primitive. Then α also controls the motion along e_2. Let e_3, e_4 be the other two edges incident to v; β controls the motion along the edges. When the robot computes σ_{e_1}, it also records $x = \tau_v(\alpha, \sigma_{e_1})$ and $y = \tau_v(\beta, \sigma_{e_1})$.[9]

At some point in the future, the robot returns to v. If it arrives via e_1 or e_2, then the gap it has been following is α; it can identify σ_{e_1} as the gap γ such that $\tau_v(\alpha, \gamma) = x$. If it arrives via e_3 or e_4, then the gap it has been following is β; it can identify σ_{e_1} as the gap γ such that $\tau_v(\beta, \gamma) = y$. See Figure 19.

Since the future identification problem has been solved, it is easy to compute the gap sensor distance between stable gaps. In particular, whenever the robot determines that some gap in $gaps(v)$ is stable for some edge incident to v, it can identify the other stable gaps in $gaps(v)$, then compute and record the gap sensor distance between each of the old stable gaps and the new stable gap.

[9]From the robot's perspective, α is the proximity pair which it has been following; β is the other proximity pair.

In the above discussion, we have ignored a minor difficulty. Suppose that the robot is oriented at 0 radians when it arrives at v, and it detects a proximity pair composed of α, α', where α' is the rightmost gap. If the robot returns to v and is oriented at π radians, then it will perceive α as the rightmost gap. If the robot tries to find a stable gap whose distance from the proximity pair is recorded as x, then it will find the wrong gap, since x is the gap sensor distance from α', not α. This problem is easily overcome: when computing gap sensor distances, the two gaps in a proximity pair should be counted as a single gap, not as two.

Lemma 4.11 *If the robot is at v_0, with pursuit status B_{v_0}, and it can identify $\sigma_{v_0v_1}$, then the robot can compute the pursuit status B_{v_m} that would result from executing the walk $v_0v_1, \ldots, v_{m-1}v_m$.*

Proof: To compute B_{v_m}, the robot need only execute the following recursive process. The current pursuit status B_{v_0} is written with the label for $\sigma_{v_0v_1}$ first. Using information about the events which occur along v_0v_1, and the distance measures $\tau_{v_0}(\sigma_{v_0v_1}, \gamma)$ for all $\gamma \in \mathit{gaps}(v_0)$, the robot computes the changes in the pursuit status, resulting in B_{v_1}.

Now, the robot rotates the pursuit status B_{v_1} by $\tau_{v_1}(\sigma_{v_0v_1}, \sigma_{v_1v_2})$ gaps, in the clockwise direction. The robot then recursively computes the pursuit status that results from the walk $v_1v_2, \ldots, v_{m-1}v_m$, resulting in the correct value of B_{v_m}. Correctness follows from Lemmas 4.10 and the solution to the future identification problem. ■

Lemma 4.11 yields a remarkable result: the gap sensor can replace a perfect compass. We think of a stable gap as a kind of "local North pole," providing a fixed point against which all other gaps can be tracked. Gap sensor distances replace angles, allowing the robot to measure distances between gaps while relying on easily measured (or sensed) quantities.

Theorem 4.12 *If it is possible to clear R, then a pursuer robot that executes the motion strategy presented in Section 3 will successfully clear R.*

Proof: Since it is possible to clear R, then by Theorem 4.4 it is possible to clear R by moving only through R'. As a result of Theorem 4.9, g_n is constructed correctly. Since g_n can be correctly explored using the algorithm described above, then by Lemma 4.11, g_s can be correctly extended using g_n. As a result, g_s can be searched for a path which clears R. ■

If the robot is placed in a polygonal environment, the size of G_n is $O(n^3)$, in which n is the number of edges [15]. For environments that can be represented piecewise algebraically, the bounds in [19] apply directly to G_n. The running time is the same as that in [14, 19], which is exponential in the number of appear lines due to the worst-case exponential number of possible statuses at a single vertex of G_n. However, the worst case rarely arises in practice, and reasonable computational performance has been observed in our implementations [20]. Furthermore, it may be possible to exploit the recent results of [25] to dramatically reduce the number of statuses that need to be considered. This remains an open topic for further research.

5 Simulation Results

The algorithm has been implemented in C++ on a 1000MHz Linux PC. In the implementation, we assumed that the robot is placed in an environment that is bounded by a simple polygon. All motions, however, are determined using information only from a simulated gap sensor. The algorithm successfully computed results for all examples. For examples that have no solution, it correctly reports failure in a few seconds.

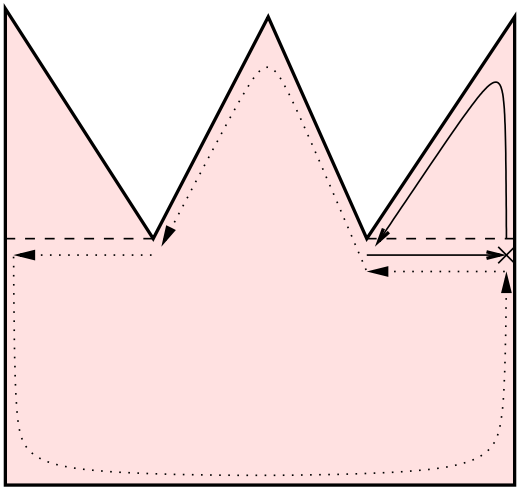


Figure 20: A simple computed example.

A very simple example, shown in Figure 20, starts out with the robot at point marked "X" facing north. The gap sensor reveals that there are three ways to go: following the wall forwards or backwards, and moving towards the proximity pair to its left. The exploration starts by going forward, and issuing left turns until the "X" is reached again (see the solid lines with arrows in Figure 20). Gap events are monitored along the way, and the result is a face in g_n; the first pass of envisioning runs (yielding no solution, yet). There are now two unknown edges in g_n: one from "X" going down, and one from the junction to the left of "X" going left. The robot continues by exploring the face containing those edges (dotted lines in figure), and in doing so clears the region. A second pass of envisioning would now find a solution, but this is not necessary as the problem has already been solved; the pursuit status is $\langle 0 \cdots 0 \rangle$.

Two other examples are shown in Figures 21 and 22. The example in Figure 21 has 15 faces, and has been solved by the algorithm in 1 second, with a motion strategy consisting of 36 primitive motions. The second has 61 faces, took 23 seconds to compute, and has a motion strategy consisting of 183 primitive motions

We refer to Figure 22 as the "eagle" example [26]. In this example, the robot is required to recontaminate the top portion of the environment multiple times, for the following reason. The robot must visit the "toes" at the ends of the "eagle's feet" (the six small triangular regions towards the bottom of the figure). To clear a toe, the robot must travel to the end of a foot, but in traveling there, the robot crosses a merge line (a line tangent to the entrance to the foot and the entrance to the beak). As the robot crosses the merge line, it loses sight of the "beak" (the triangular region at the top of the figure). The beak becomes recontaminated; therefore, each time the robot clears a toe, it must immediately revisit the beak.

If the robot starts on the right side of the figure, outside both the beak and any feet, then a motion strategy that clears the region can be described as follows. The robot travels to the bottom of the rightmost foot; then it travels to the beak, without crossing any merge lines; then it visits the next foot to the left; then it visits the beak, again without crossing any merge lines. The robot continues in this way until the final toe, and then the beak, are cleared.

Based on several examples, we have found that the time for computing a solution is within a few seconds, and is much less than the time that would be required for a pursuer robot to move in a typical environment. As a result, we believe the algorithm is practical for use in real-time pursuit evasion problems.

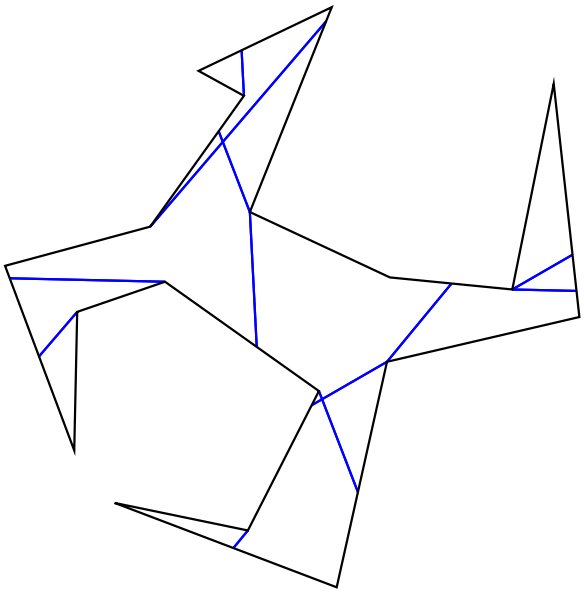


Figure 21: A more-challenging environment. Only the faces are shown.

6 Discussion

We have presented and implemented a complete algorithm which enables the limited-sensing pursuer to solve any problem that could have been solved by a pursuer that has a complete map and perfect navigation capabilities (under some general position assumptions). The motion strategy relies entirely on wall-following capability and the ability to move in the direction of proximity pairs, which are based on its target detection sensor. Although in its current form, the algorithm is not ready to implement in a mobile robot, it removes many unnecessary components from previous, idealized models. The gap sensor model has been validated on real mobile robot hardware in [32]; however, the motions generated by our proposed algorithm are very complicated. It should be possible to generate much simpler motions, but this remains a topic of future research. In any case, the pursuit-evasion strategies determined by our algorithm do not depend on complicated environment models, which are often difficult or impossible to construct with great accuracy. Therefore, we believe this work will enable the development of low-cost robots that can complete pursuit-evasion tasks with a high degree of robustness and autonomy. We imagine that a mobile robot might one day be taken "from the box," be placed in an unknown environment, and be able to systematically search for all moving targets.

The completion of this work raises many interesting and challenging questions. In the appendix, we consider the constraints placed on the environment by the ϵ control and δ sensing requirements. It might be possible to weaken the ϵ control precision requirement, thus weakening the corresponding constraints on the environment. Also, the general position requirement might not be necessary. For the exploration of a face, it might be possible to detect a cycle by simply examining the gap events, at least for some class of environments. Further, we make no claims that the number of primitive motions executed by the robot is optimal; constructing a competitive ratio bound for the on-line problem remains a considerable challenge.

It is also interesting to consider extensions to pursuers that have other evader detection and sensing models. For example, the pursuer might have limited distance perception, or might carry a finite collection of beams for detecting evaders. Also, it might be possible to replace omnidirectional vision in our algorithm with two-flashlight vision, due to the results in [29, 25]. To handle more-complicated environments, it will

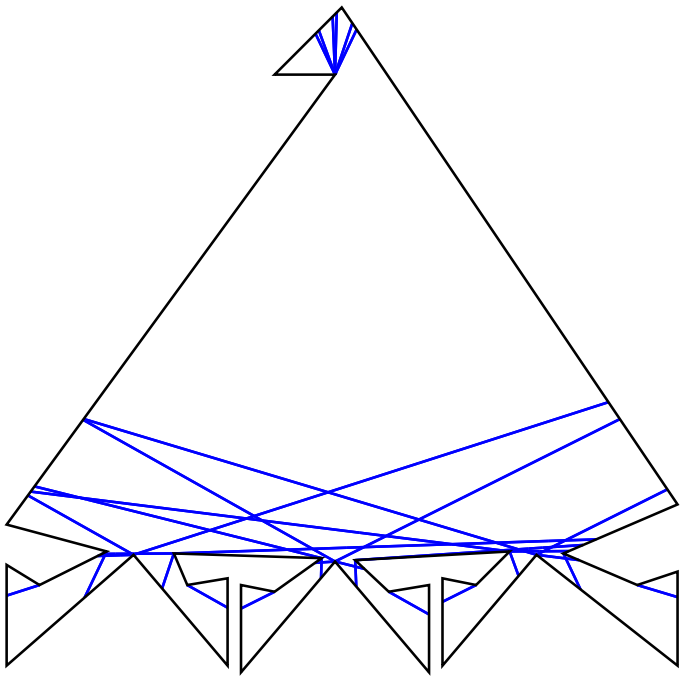


Figure 22: An example that requires recontaminations.

be necessary to coordinate the efforts of multiple pursuers, which is a considerable challenge (finding the minimum number of pursuers is NP-hard for the ideal information case in a polygonal environment [14]).

Our focus was primarily on ensuring that our algorithm yields solutions for as many environments as possible, as opposed to trying to optimize the distance traveled by the robot. Obtaining a stronger result, such as a competitive ratio [24], as is often considered in on-line algorithms, remains an open problem. In fact, finding shortest-path solutions to pursuit-evasion problems with complete maps is also an open problem.

Acknowledgments

We are grateful for the funding provided in part by NSF CAREER Award IRI-9875304 (LaValle), NSF IRI-0116592, and an NSF REU, all of which are granted by the Robotics and Human Augmentation program. We are also grateful for funding provided by the Office of Naval Research (ONR grant N00014-02-1-0488 from the Autonomous Systems Program). We thank Boris Simov for the concept of "countries" and "provinces" in describing the motivation for the exploration strategy. We thank Elon Rimon for an encouraging discussion on developing a minimal model. We also thank Rafael Murrieta-Cid for helpful advice on sensing assumptions in true mobile robot systems, and Benjamin Tovar for conducting mobile robot experiments that use the gap sensor model.

Appendix

In Section 2.4, we assume that the robot is within ϵ distance of a merge line (on the split side) if and only if the gaps corresponding to the two unseen regions are within δ angular distance in the gap sensor. Although this presents no theoretical difficulties, there are some practical issues and limitations regarding this assumption. The difficulties arise from the lack of *any* distance information available to the robot. Even coarse distance information can be used to alleviate this difficulty.

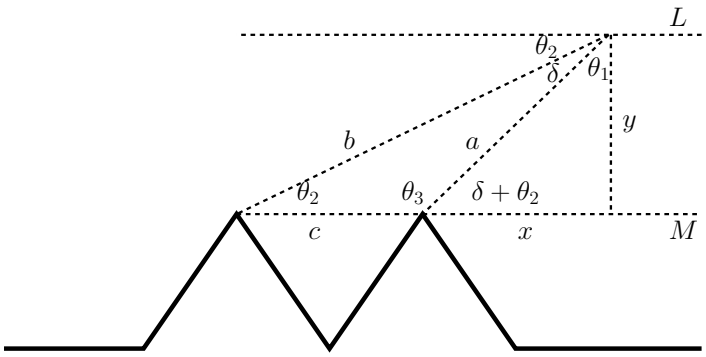


Figure 23: Two points of tangency are c distance apart; the angular distance between the corresponding gaps is δ. The line L is parallel to the merge line M, and is used to construct parallel angles.

Suppose there are two points of tangency which are c distance apart, as in Figure 23. We wish to find the region in which the corresponding gaps are δ angular distance apart. Suppose the robot is distance x to the right and y above the rightmost point of tangency.

We can compute δ as a function of x, y and c using simple geometry:

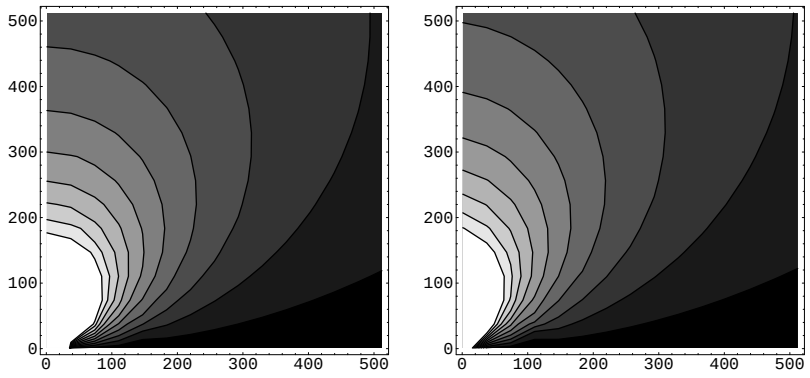
$$\begin{aligned}
\frac{c}{\sin\delta} &= \frac{b}{\sin\theta_3} \\
\theta_1 &= \frac{\pi}{2} - (\delta + \theta_2) \\
\theta_3 &= \pi - (\delta + \theta_2) \\
&= \theta_1 + \frac{\pi}{2} \\
\theta_1 &= \tan^{-1}\frac{x}{y} \\
\delta &= \sin^{-1}\frac{c\sin\theta_3}{b} \\
&= \sin^{-1}\frac{c\sin\theta_3}{\sqrt{y^2+(x+c)^2}} \\
&= \sin^{-1}\frac{c\sin(\theta_1+\frac{\pi}{2})}{\sqrt{y^2+(x+c)^2}} \\
\delta &= \sin^{-1}\frac{c\sin\left(\left(\tan^{-1}\frac{x}{y}\right)+\frac{\pi}{2}\right)}{\sqrt{y^2+(x+c)^2}} \qquad (1)
\end{aligned}$$

Figure 24 shows contour maps of Equation 1. Several salient features emerge:

1. If we let the point of rightmost tangency in Figure 23 be the origin, then a single contour curve representing some constant δ is the image of a path π such that the gaps are exactly angular distance δ apart whenever the robot is at $\pi(t)$, for all t. They are less than δ apart whenever the robot is in the region between $\pi(t)$ and M, for some t.

2. As c grows larger and x and y remain constant, δ grows larger; that is, for all x, y, $\frac{\partial\delta}{\partial c} > 0$.

3. As c grows larger and x and y remain constant, each contour curve grows "flatter"; that is, for all x, y, $\frac{\partial^2\delta}{\partial c^2} < 0$.



(a) (b)

Figure 24: A contour map of Equation 1, with (a) $c = 2$ and (b) $c = 64$. In both graphs, $1 \leq x, y \leq 512$. Note that darker regions indicate lower values of δ, and that regions with similar shading in (a) and (b) do not necessarily have similar values of δ.

As a result of these observations, we can now make some observations about the required value of δ. Let c_0 be the largest distance between points of bitangency, and let x_0 be the length of the largest merge line (recall that a merge line is a line from a point of bitangency to ∂R). Let ϵ be the robot's maximal control error, as described in Section 2.2. Then δ can be computed according to Equation 1, with $c = c_0$, $x = x_0$, and $y = \epsilon$.

A few small difficulties remain in the guarantee that the robot will be within ϵ distance of M if and only if the two gaps in a proximity pair corresponding to M are δ distance apart in the gap sensor. First, we have not considered this issue for the case of curved environments. In this case, c is not a constant, but a function of x and y; a similar analysis could be performed. Second, we note that as the robot gets closer to the origin (the rightmost point of bitangency), it must be closer and closer to the x-axis (the line M) in order to see the gaps within δ angular distance. We believe that mistakes in these small regions can be corrected using coarse odometry. Finally, we have noted that it is necessary to know c_0 and x_0 in order to compute δ.

Our original model is therefore slightly weakened; we require that the robot (or its designer) has some knowledge about the environment. However, c_0 and x_0 may be approximated, and in any case retrieving them reliably is much easier than retrieving a map of R. We believe that the required relationship between ϵ and δ does not significantly impede the practical implementation of this work; however, experimental evaluation in a real system would be the ultimate test.

References

[1] E. U. Acar and H. Choset. Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and voronoi diagrams. In *Proc. of IEEE IROS, Int'l Conference on Intelligent Robots and Systems*, 2001.

[2] E. U. Acar and H. Choset. Robust sensor-based coverage of unstructured environments. In *Proc. of IEEE IROS, Int'l Conference on Intelligent Robots and Systems*, 2001.

[3] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Sensorless parts feeding with a one joint robot. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 229–237. A K Peters, Wellesley, MA, 1997.

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

[5] 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.

[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] D. Crass, I. Suzuki, and M. Yamashita. Searching for a mobile intruder in a corridor – the open edge variant of the polygon search problem. *Int. J. Comput. Geom. & Appl.*, 5(4):397–412, 1995.

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

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

[10] B. R. Donald and J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. In *IEEE Int. Conf. Robot. & Autom.*, pages 190–197, Sacramento, CA, April 1991.

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

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

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

[14] 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.

[15] 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.

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

[17] I. Kamon, E. Rivlin, and E. Rimon. Range-sensor based navigation in three dimensions. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[18] K. N. Kutulakos, C. R. Dyer, and V. J. Lumelsky. Provable strategies for vision-guided exploration in three dimensions. In *IEEE Int. Conf. Robot. & Autom.*, pages 1365–1371, 1994.

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

[21] J.-H. Lee, S. Y. Shin, and K.-Y. Chwa. Visibility-based pursuit-evasions in a polygonal room with a door. In *Proc. ACM Symp. on Comp. Geom.*, 1999.

[22] V. Lumelsky and S. Tiwari. An algorithm for maze searching with azimuth input. In *IEEE Int. Conf. Robot. & Autom.*, pages 111–116, 1994.

[23] 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.

[24] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 322–333, 1988.

[25] S.-M. Park, J.-H. Lee, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searche r. *Lecture Notes in Computer Science*, 2076:456–??, 2001.

[26] 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.

[27] A. M. Shkel and V. J. Lumelsky. Incorporating body dynamics into sensor-based motion planning: The maximum turn strategy. *IEEE Trans. Robot. & Autom.*, 13(6):873–880, December 1997.

[28] B. Simov, G. Slutzki, and S. M. LaValle. Pursuit-evasion using beam detection. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2000.

[29] I. Suzuki, Y. Tazoe, M. Yamashita, and T. Kameda. Searching a polygonal region from the boundary. *International Journal on Computational Geometry and Applications*, 11(5):529–553, 2001.

[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] I. Suzuki, M. Yamashita, H. Umemoto, and T. Kameda. Bushiness and a tight worst-case upper bound on the search number of a simple polygon. *Information Processing Letters*, 66:49–52, 1998.

[32] B. Tovar, S. M. LaValle, and R. Murrieta. Optimal navigation and object finding without geometric maps or loca lization. In *IEEE Int. Conf. Robot. & Autom.*, 2003.

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