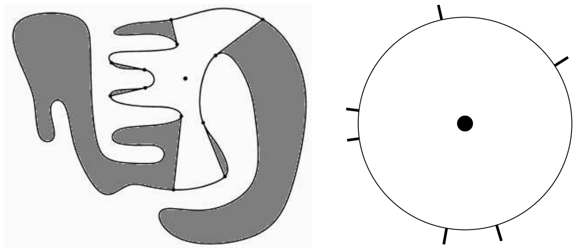
A Pursuit-Evasion BUG Algorithm

Stjepan Rajko Steven M. LaValle
Dept. of Computer Science
Iowa State University
Ames, IA 50011 USA
{srajko, lavalle}@cs.iastate.edu

Abstract

We consider the problem of searching for an unpredictable moving target, using a robot that lacks a map of the environment, lacks the ability to construct a map, and has imperfect navigation ability. We present a complete algorithm, which yields a motion strategy for the robot that guarantees the elusive target will be detected, if such a strategy exists. It is assumed that the robot has an omnidirectional sensing device that is used to detect moving targets and also discontinuities in depth data in a 2D environment. We also show that the robot has the same problem-solving power as a robot that has a complete map and perfect navigation abilities. The algorithm has been implemented in simulation, and some examples are shown.



(a) (b)

Figure 1: Discontinuities in depth measurements partition the set of viewing directions. Each discontinuity could hide an evader.

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.

This paper presents the first approach to visibility-based pursuit-evasion that does not require a complete map of the environment to be specified a priori. Instead, we adopt the minimalist philosophy presented in [12] 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 [3, 7, 8, 15]. This mind set inspires our current work, which applies these principles to pursuit-evasion problems, resulting in a pursuit-evasion BUG algorithm, or PE-BUG for short. The primary benefit of this approach is that the sensing requirements of the robot are greatly reduced. This can lead to lower-cost solutions that ultimately may achieve greater robustness due to less dependencies on accurate, complicated models.

The pursuit-evasion problem, however, is considerably more complex than basic navigation. To indicate the difficulty of the problem, we briefly summarize previous solutions, which apply to the ideal case in which the environment is known exactly, and the robot has perfect motion and sensing capabilities. Visibility-based pursuit-evasion in a polygonal environment was first considered in [17], in which incomplete algorithms were given for cases in which the pursuer that 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 [10]. Solutions to the case in which the pursuer has one or more detection beams are considered in [4, 11, 16, 18], for various types of polygons. A pursuit-evasion algorithm for curved environments was presented in [9]. In [5], both optimal and approximation algorithms were presented for the case of a chain of pursuers that maintain mutual pairwise visibility in a polygonal environment. Such challenges with the ideal pursuit-evasion problem might cause hopes of developing a BUG algorithm dwindle; however, we present an algorithm which enables the PE-BUG to solve any problem that could be solved with an ideal pursuer, which has a complete map and perfect navigation abilities.

2 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 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. Let $e_i(t) \in R$ denote the position of the i^{th} evader at time $t \geq 0$. It is assumed that $e_i : [0, \infty) \rightarrow R$ is a continuous function, and each evader is capable of moving at a finite, unbounded speed. 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. Let $V(x)$ be called the *visibility region.* Let $\gamma(t)$ denote the position of the pursuer at time $t \geq 0$; the pursuer also moves continuously. If at any time t, $e_i(t) \in V(\gamma(t))$, we say that the i^{th} evader is detected (and eliminated). In the spirit of [14], 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 will be referred to as *recontaminated.* It is assumed that the pursuer does not know the starting position, $e_i(0)$, or the path, e_i, of any evader. Initially, each evader could be anywhere in R that is not visible from $\gamma(0)$. The task is to ensure that the pursuer will follow a path that causes all of R to become cleared.

We now define a simple pursuer model that is inspired by BUG algorithms and related paradigms. We make no assumptions on the pursuer's initial knowledge of R, or its capability to construct a map, obtain accurate depth measurements, and perform localization. In the interest of robustness, it might be advantageous to avoid such constructions even if the pursuer is capable. Finally, it is assumed that the pursuer can execute omnidirectional motions.

Some sensing capability is required, of course, to solve the problem. Consider Figure 1.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 1.b, which gives the location of discontinuities in depth information. In a sense, Figure 1.b indicates the way the world appears to the pursuer at all times. Note that each gap corresponds to a connected portion of R that is not visible to the pursuer. The precise distances to the walls may be unknown; however, it is assumed that the pursuer has a kind of edge detector that can detect each of the discontinuities, and return their direction relative to the pursuer's heading. 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.

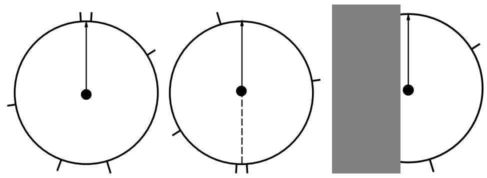


Figure 2: Three situations: a) moving towards a proximity pair; b) moving away from a proximity pair; c) moving along a wall.

Given the sensing and knowledge limitations of the pursuer, special care must be given in specifying a motion strategy. As in the case of BUG algorithms, we assume that the pursuer is capable of following arbitrarily close to walls. One additional kind of motion is assumed, and is based entirely on information from the gap sensor. Consider two gaps, a_1 and a_2, which are observed by the gap sensor, and a fixed small constant, $\epsilon > 0$. While the pursuer moves, it is possible that $|\theta(a_i) - \theta(a_j)| < \epsilon$, as shown in the left of Figure 2. We refer to this situation as a *proximity pair.* Another proximity pair can occur if one gap, a, splits into two gaps, a_1 and a_2. Let $\theta(a_i, a_j)$ denote the average direction between a_i and a_j. If the pursuer is commanded to move in directions orthogonal to $\theta(a_i, a_j)$ (i.e., either $\theta(a_i, a_j) + \frac{\pi}{2}$ or $\theta(a_i, a_j) - \frac{\pi}{2}$), the gaps will either quickly diverge or quickly merge. If the pursuer is commanded to move in the direction of $\theta(a_i, a_j)$ or $\theta(a_i, a_j) + \pi$, the gaps will remain close, but will not merge. We will assume that a motion command can be issued to the purser that keeps the gaps within some small threshold, ϵ, of each other, but does not allow them to merge (no dynamics are modeled). This can be accomplished either by moving directly toward gaps a_i and a_j, or by moving directly away. Small errors in control can be corrected by using feedback from the gap sensor to vary the commanded direction.

For convenience, any pursuer motion strategy will be described in terms of simple primitives, each of which is terminated by a *junction.* If the pursuer is moving along a wall, a junction is encountered when a proximity pair arises. If the pursuer is moving in direction θ or $\theta + \pi$ for a pair of gaps, then a junction is encountered when a new proximity pair arises, or the wall is approached.

At each junction, one of four 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 simply follows the wall to the right. Otherwise, the pursuer moves right by a commanded motion either in the direction towards or against the new proximity pair.

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

We do make a general position assumption, which is stated in Section 4, that ensures only one proximity pair is encountered at a time.

Since the pursuer does not have complete information, its motion strategy must depend on information gathered during execution. Section 3 presents an algorithm which tells the robot which primitive motion to execute, each time it arrives at a junction. If the problem can be successfully solved, the motion strategy guarantees that each evader will be detected.

3 The PE-BUG Algorithm

The pursuer has the task of detecting all of the evaders, but it cannot achieve this without accumulating some knowledge about its environment. Therefore, the algorithm involves two interleaved processes: *exploring* the environment, and *envisioning* movements that would lead to the detection of all evaders. The exploring process involves movement of the robot, and the construction of a topological representation of the environment as a graph based on previously-executed primitive motions. During exploration, the status of the pursuit is also maintained (i.e., where might evaders be hiding?). The envisioning process uses the representation of the environment to determine if the pursuit-evasion task can be solved using the representation of the environment so far. If so, then the pursuer executes motions that solve the task. If not, then more exploring is required, and the process continues. It is possible to do all exploration first, followed by envisioning; however, we choose to interleave the processes for efficiency reasons. The pursuer may be able to solve the problem without exploring the whole environment.

The navigation graph First, consider a topological environment representation that is based on the perspective of the pursuer and its gap sensor. We define the *navigation graph*, G_n, by using only the primitive motions and junctions. Each junction corresponds to a vertex in G_n. An edge exists between two vertices if one junction can be reached from another by a single motion primitive. Initially, G_n is unknown to the pursuer. For clarity, the notation g_n will be used to represent the portion of G_n that is known to the pursuer at a given time.

The pursuit status As the pursuer moves in its environment, it uses its gap sensor to keep track of the status of the pursuit. Recall from Figure 1, that each gap corresponds to a connected region of R that is not visible. Initially, any hidden regions might contain an evader. Eventually, all hidden regions will be cleared. For each hidden region, a binary label can be used as: "1" to indicate that the region is contaminated, and "0" to indicate it is cleared. Let the *status*, B, denote a sequence of binary labels, which indicates if each of the hidden regions is clear or contaminated, while the pursuer is at a fixed location.

Recall that the pursuer has the ability to track gaps as it moves. As gaps change, the pursuer must correctly update the status. There are four possible ways in which the gaps change [9]: 1) a new gap appears; 2) an existing gap disappears; 3) two or more gaps merge into one; 4) a gap splits into two or more gaps. Suppose that one of these events occurred, and B must be revised to B' after considering the gap change. If a gap disappears, a bit simply disappears when going from B to B'. If a gap appears, then it always receives a "0" label, because it corresponds to a hidden region that was previously visible before the event, and therefore contains no evaders. If several gaps merge into one, then the corresponding bit in B will be the logical OR of the corresponding bits in B'. This is correct because one contaminated region could spread to other regions. If one gap splits into several, the corresponding bits in B' will each receive the label of the corresponding bit in B.

Envisioning a solution Suppose for a moment that G_n is completely known. Using this, a directed *status graph*, G_s, can be constructed that yields all possible situations that could be obtained, as a result of sequences of primitive motions. For each vertex, $v \in G_n$, a set of vertices of G_s can be defined. Recall that each vertex in G_n corresponds to a junction that is visited by the pursuer. There is one vertex in G_s for each possible sequence, B, of binary labels that could be applied to the gaps. Thus, there are 2^k vertices in G_s per $v \in G_n$, in which k is the number of gaps identified by the gap sensor. Let v_1 and v_2 be two vertices in G_s, and let v_1' and v_2' be their corresponding vertices in G_n. A directed edge in G_s exists from v_1 to v_2 if and only if: 1) there is an edge in G_n from v_1' to v_2', and 2) the status represented by v_1 would be transformed into the status represented by v_2 when the pursuer moves from v_1' to v_2'. Given the current junction and binary sequence, B, there is a corresponding vertex in G_s. If there is a path in G_s from this vertex to a vertex for which $B = \langle 0\ 0\ \cdots 0 \rangle$, then a sequence of primitive motions exists that can be applied to solve the pursuit problem. Note that the same junction may have to be visited multiple times, but each time the status could be different, resulting in a different vertex in G_s.

It was stated previously that G_n is not known to the pursuer; however, some subgraph g_n is revealed as the environment is explored. Let g_s refer to the subgraph of the G_s that can be inferred by only applying binary labels at junctions that correspond to the vertices in g_n. As the robot explores the environment, the graph g_n will be extended. The envisioning process involves extending g_s to include vertices that correspond to the new junctions, and then searching g_s for a path from the current vertex in g_s to a solution vertex (with all labels "0"). If a solution is found, the robot immediately executes the solution. Otherwise, more exploration is performed.

Exploring a face Note that G_n is a planar, undirected graph, because it is derived from motions in a planar environment and each motion is reversible. First, we consider the problem of exploring a single face of G_n. Suppose that the pursuer is at a junction, and hence a corresponding vertex of G_n. The following sequence of motion primitives, $\langle L, L, \ldots, L\rangle$, are issued until the same junction is reached again. [1] Recall that during the execution of the primitive motions, the status must be maintained.

The global exploration strategy The next step is to specify the order in which faces are explored. This exploration must occur in a systematic way to ensure that whenever a face is encountered a second time, the pursuer is able to determine that it is not a new face. Suppose the pursuer starts its exploration at a wall; if the pursuer is instead in the interior of R, then it can easily move until it hits a wall. At any given time in the exploration, we refer to the *unexplored faces* and the *explored faces*. Each iteration will explore a set of faces. In the first iteration, a single face is explored, to obtain g_n. For any point in the search, let the *frontier vertices* refer to the set of all vertices in g_n that are vertices of unexplored faces, called the *frontier faces*. In each iteration, the frontier faces are explored in order, from wall to wall, as shown in Figure 3. In alternating iterations, they are explored from left to right, and then from right to left, to reduce the number of wasted motions. In some iterations, there will be more than two frontier vertices that correspond to a wall, which corresponds to a split in the environment. In these cases, each part of the environment is explored recursively. After one part is completed, the pursuer follows the wall to reach the next part.

After each face is explored, an envisioning operation is performed to determine whether the pursuer has learned enough of the environment to solve the problem. If so, then the exploration phase is terminated, and the pursuer executes the computed motion primitives that lead to the solution.

[1]There are several ways of detecting the return to a junction along the face boundary. It can drop a marker in the environment, and detect it upon return (similar to the use of pebbles in on-line exploration [1]). We have also considered methods involving the use of a poor odometer, compass (e.g., [2]), or the analysis of the gap sensor and motion command history. The completness of such methods, however, remains to be proven.

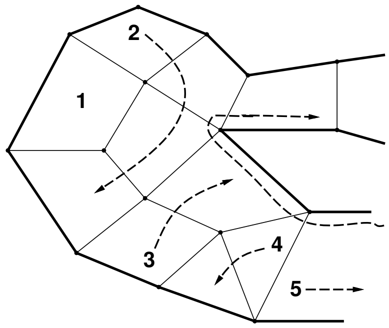


Figure 3: The exploration strategy ensures that if a junction is revisited, it is not misinterpreted as a newly-encountered junction.

4 Algorithm Analysis

This section briefly sketches the completeness arguments for the PE-BUG algorithm (the proofs and definitions are too lengthy to include in detail), and motivates the choices of motion primitives and exploration strategies. Although the pursuer cannot "see" the environment, we must consider the environment to analyze the problem. Recall that R is bounded by a continuous, piecewise-smooth curve. Consider the inflections of the boundary, and note that when such an inflection is crossed a gap will either appear or disappear. We term such a line an appear line.

Similarly, consider the bitangents induced by the boundary (i.e. all line segments in R that are tangent to the boundary at two points). Note that crossing the 'outer' portions of the bitangent causes two gaps to merge or split. Therefore, we term such a line a merge line.

The set of all merge lines induce a partition of R into cells that have walls and/or merge lines as their boundaries. Primitive motions towards or against a proximity pair correspond to following a merge line closely, on the side where the gaps are split. The general position assumption can now be stated: no more than two appear and/or merge lines intersect at the same point in the interior or R, and there are no parallel, overlapping merge lines. We are currently investigating the removal of this restriction, and allowing degenerate environments.

G_n can be considered the graph induced by the pursuer following the boundary and merge lines "close enough." We define this "close enough" distance to be ϵ_{max}, and define R' to be the one dimensional subset of R that includes all points whose minimum distance from the wall and merge lines (that have the point on the split side) is ϵ_{max}. Note that the distance at which the pursuer follows lines can fluctuate as long as it stays within ϵ_{max}; however, for convenience in the following discussion, we assume that the pursuer moves in the skeleton R'.

We now consider several lemmas and theorems, culminating in Theorem 4.7, which establishes the completeness of the algorithm and that the pursuer has the same power as a pursuer with a complete map and perfect navigation abilities.

The following lemma is straightforward to establish:

Lemma 4.1 *If there exists a solution path, $\gamma : [0,1] \to R$, then there exists another solution path γ' such that $\gamma'(0) \in R'$ and $\gamma'(1) \in R'$.*

We now use information space concepts, which are defined formally in [6, 9]. The information state represents set of all places in which an evader could be hiding, and is specified in the present context by identifying the gaps that appear in the gap sensor, and assigning a binary label to each. The label represents clear or contaminated, as in the case of the status from Section 3, which can be considered as the information states at junctions. Due to appear lines, however, a sequence of information states are traversed during the execution of a primitive motion. The present definition of information states can be reduced to more basic information states as described in [6] by defining equivalence classes; however, we forego this discussion in this paper.

A partial ordering can be defined on the set of information states by considering one information state, η_1 to *dominate* another, η_2 if each involves the same set of gaps, and for each gap the label for η_1 is "0" if the label for η_2 is "0".

The following lemma is crucial to establishing that the pursuer can move along R' and have the same problem-solving ability as if it moved anywhere within R.

Lemma 4.2 *For any path $\gamma : [t_0, t_1] \to R$, such that $\gamma(t) \in R'$ if and only if $t \in \{t_0, t_1\}$, then there exists a path $\gamma' : [t_0, t_1] \to R'$ such that $\gamma'(t_0) = \gamma(t_o), \gamma'(t_1) = \gamma(t_1)$, and the information state at $\gamma'(t_1)$ dominates the information state at $\gamma(t_1)$ (this means that executing γ will clear all gaps cleared by executing γ', while not contaminating any gaps that executing γ does not contaminate).*

Sketch of Proof: Due to the nature of γ, we can see that either γ goes between R' and ∂R, or there exists a simple closed loop $L \subset R'$, such that both $\gamma(t_0)$ and $\gamma(t_1)$ are on points in L, and $\gamma(t)$ is in the interior of L for $t_0 < t < t_1$. Then we know that $\gamma(t_1)$ can be reached through R', but we still need to show that it can be done in a way that produces a dominant information state.

In the first case, the lemma follows from the fact that following ∂R closer than ϵ_{max} is equivalent to following at ϵ_{max} In the other case, first suppose that L is composed of line segments surrounding n merge lines $b_1 \ldots b_n$ enumerated clockwise, and with $\gamma(t_0)$ directly outside b_1 (there is also another case, in which $\gamma(t_0)$ is outside both b_1 and b_2, and the proof is a simple extension of the treatment given below). Recall that due to our definition of R', 'surrounding' means that if a pursuer moves from any point in L towards the inside of L, a pair of gaps will merge (recall that a cleared gap can only be contaminated if it merges with a contaminated gap). If the pursuer starts moving from $\gamma(t_0)$ along L clockwise, the first critical gap change (ignoring appear line crossings) will be a split, S_2, caused by crossing b_2, followed by a merge, M_1, caused by crossing b_1. Using the same notation, going all the way around L would cause critical gap changes $S_2, M_1, S_3, M_2, \ldots, M_n, S_1$ in precisely that order. If it were the case that the part of L we supposed was outside b_i was actually following a part of ∂R or the merge line corresponding to b_i was outside L, the pair of gap changes S_i, M_i would simply never occur. Note that throughout this sequence of gap changes, the only time information can be lost is at M_1, if it is triggered. This is because prior to every other merge, there was already a split, and between the two there can be no gap changes that will contaminate any of the newly-split gaps, which means that the merged gap will not be contaminated unless it was contaminated before the split.

There are two cases. First, if γ triggers M_1 then the pursuer can start from $\gamma(t_0)$, safely go all the way around L, and then continue around L to reach $\gamma(t_1)$. Going all the way around L, the pursuer will cross all appear lines that intersect L, and therefore clear all gaps that executing γ would have cleared. Also, if doing so caused contamination upon crossing M_1, then executing γ would have also caused the same contamination. Second, if γ does not trigger M_1, then the pursuer can start from $\gamma(t_0)$, visit all points on L that can be reached without triggering M_1, and then continue around L to $\gamma(t_1)$. This path will cross all appear lines that γ crosses, and therefore clear all gaps executing γ would have cleared. In either case, upon reaching $\gamma(t_1)$ we have ensured that a dominant information state will be obtained. ∎

Lemmas 4.1 and Lemmas 4.2 can be used to establish the following, which indicates the power of the PE-BUG:

Theorem 4.3 *A robot capable of navigating only through G_n can clear any clearable region R.*

It remains to show that G_n can be determined using the exploration strategy given in Section 3, and that it can be successfully searched for a solution if one exists. Each gap exists because of an appear line, and the key difficulty is ensuring that whenever the

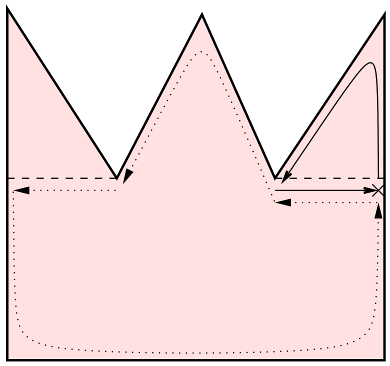


Figure 4: A simple computed example.

purser sees a gap for the second time, it is not confused as a new gap. In other words, there are different places during navigation in which gaps detected by the gap sensor share the same appear line. This means that they correspond to the same hidden region of R, and it is essential for the pursuer to recognize this to have the same power as the pursuer with a complete map. Once the correct correspondences between gaps is determined, the proper information state can be computed by the pursuer. This leads to correct computations of the status, which enables to pursuer to correctly execute the envisioning phase.

The exploration strategy shown in Figure 3 was designed to satisfy the following lemma:

Lemma 4.4 *During the global exploration, the boundary of every newly-explored face contains one chain of edges in g_n, and one chain of new edges.*

Along the current face, the edges in g_n border faces known to the pursuer, and the remaining edges identify unexplored faces. This avoids the problem of having to deciding for each edge that borders an unknown face whether or not it already exists in g_n.

Lemma 4.5 *During the exploration of a face, gaps that share the same appear line can be identified, or all gaps will be cleared.*

Lemma 4.5 ensures that gaps are identified correctly within a face, and Lemma 4.4 can be used to ensure that corresponding appear lines between gaps that appear in different faces are correctly identified. This leads to the following theorem:

Theorem 4.6 *All correspondences between gaps are determined correctly in the exploration of R.*

This implies that information states are correctly handled during the entire exploration. Theorems 4.3 and 4.6 can be used in establishing the following following theorem, which concludes this section.

Theorem 4.7 *The algorithm correctly solves any pursuit-evasion problem that could be solved by a pursuer that uses a complete algorithm on an environment with a complete map and perfect navigation abilities.*

5 Implementation and Simulations

The algorithm has been implemented in C++ using the LEDA library, on a 1000MHz Linux PC. In the implementation, we assumed that the pursuer 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 several examples. A very simple example, shown in Figure 4, starts out with the robot at point marked “X” facing north. It has no map of the environment, but its sensor information 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 arrowed lines in figure). Gap events are monitored along the way, and the result is a face in g_n with edges parametrized to reflect gap events, which allows the first pass of envisioning to run (which yields no solution, at this point). 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.

Two other examples are shown in Figure 5. The first example has 15 faces, and has been solved by the algorithm in 1 second. The second has 61 faces, and took 13 seconds to compute. In the second example, the pursuer is required to recontaminate the top portion of the environment multiple times. These motions were determined by the envisioning phase. We believe the execution times can be substantially reduced because at the present time our implementation is poorly optimized. In an implementation on a real robot, time must be allotted for the actual robot motions as it explores the environment, which also enables more computation time in solving the pursuit evasion problem.

6 Conclusions

We have presented and implemented a complete algorithm for the PE-BUG, which enables it to solve any problem that could have been solved by a pursuer that has a complete map and perfect navigation capabilities. The PE-BUG 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,

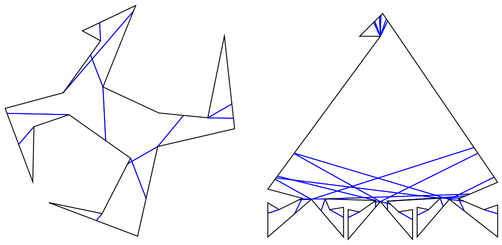


Figure 5: Two more-challenging environments that were correctly solved by the PE-BUG. The faces are shown.

it removes many unnecessary components from previous, idealized models. We believe this will enable the development of low-cost robots that can complete pursuit-evasion tasks with a high degree of robustness and autonomy because the pursuit-evasion strategies determined by our algorithm not depend on complicated environment models, which are often difficult or impossible to construct with great accuracy. We imagine that a PE-BUG 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. For example, we make no claims that the number of primitive motions executed by the pursuer is optimal. In *competitive analysis* [13] one usually bounds the performance disadvantage of having limited information, in comparison to the ideal information case. Constructing a similar bound for the PE-BUG 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. To handle more-complicated environments, it will 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 [6]).

Acknowledgments

We are grateful for the funding provided in part by NSF CAREER Award IRI-9875304 (LaValle) and the NSF REU program. We thank Elon Rimon for an encouraging discussion.

References

[1] 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.

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

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

[4] 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.

[5] 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.

[6] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *WADS '97 Algorithms and Data Structures (Lecture Notes in Computer Science, 1272)*, pages 17–30. Springer-Verlag, Berlin, 1997.

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

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

[9] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: An extension to curved environments. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1677–1682, 1999.

[10] 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.

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

[12] 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.

[13] 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.

[14] T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Application of Graphs*, pages 426–441. Springer-Verlag, Berlin, 1976.

[15] 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.

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

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

[18] M. Yamashita, H. Unemoto, I. Suzuki, and T. Kameda. Searching for mobile intruders in a polygonal region by a group of mobile searchers. Technical Report TR-96-07-01, Dept. of Electrical Engineering and Computer Science, University of Wisconsin - Milwaukee, July 1996.