Bitbots: Simple Robots Solving Complex Tasks*

Anna Yershova† **Benjamín Tovar**† **Robert Ghrist**‡ **Steven M. LaValle**†

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

‡ Department of Mathematics
University of Illinois
Urbana, IL 61801 USA
ghrist@math.uiuc.edu

Abstract

Sensing uncertainty is a central issue in robotics. Sensor limitations often prevent accurate state estimation, and robots find themselves confronted with a complicated *information (belief) space*. In this paper we define and characterize the information spaces of very simple robots, called Bitbots, which have severe sensor limitations. While complete estimation of the robot's state is impossible, careful consideration and management of the uncertainty is presented as a search in the information space. We show that these simple robots can solve several challenging online problems, even though they can neither obtain a complete map of their environment nor exactly localize themselves. However, when placed in an unknown environment, Bitbots can build a topological representation of it and then perform pursuit-evasion (i.e., locate all moving targets inside this environment). This paper introduces Bitbots, and provides both theoretical analysis of their information spaces and simulation results.

Introduction

Information spaces (also called *belief spaces*) lie at the heart of planning for robots. The *state* of a robot system includes both the environment in which the robot is placed and its position and orientation within the environment. A common approach is to build a full map of the environment and localize the robot with respect to the map. Simultaneous localization and mapping (SLAM) has received considerable attention in recent years (Thrun *et al.* 2001; Choset & Nagatani 2001; Montemerlo *et al.* 2002) and can be considered as a method for computing the information states of sensing-intensive robot systems. Due to Bayesian uncertainty models, the information space in this case is probabilistic. Probabilistic information spaces also arise in the study of POMDPs, for which algorithms for computing optimal or near-optimal policies for small, finite state spaces are used (Kaelbling, Littman, & Cassandra 1998). One of the greatest challenges with probabilistic information spaces is that the dimension is proportional to the number of states (information spaces are infinite-dimensional for continuous state spaces).

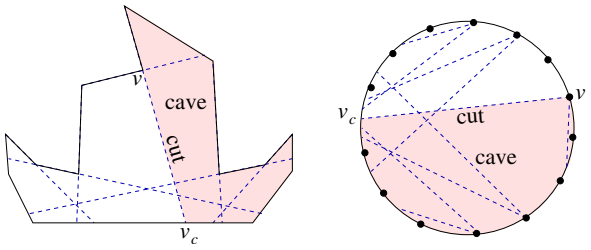

Figure 1: A polygon and all its cuts are shown on the left. For a cut $[v, v_c]$ its cave is shaded. The corresponding cut diagram is shown on the right.

Large, complicated information spaces are difficult to handle. It is therefore preferable to design robotic systems in a way that leads to simpler information spaces. One possibility that has been highly successful in the past is to use nondeterministic uncertainty models and consider worst-case analysis, as opposed to probabilistic models and expected-case analysis. For example, for the problem of visibility-based pursuit-evasion (Guibas *et al.* 1999; Gerkey, Thrun, & Gordon 2004), this approach led to exact, complete algorithms that solve the problem by partitioning the information space into equivalence classes. The resulting finite cell decomposition can be searched completely and systematically, instead of relying on approximations.

Another crucial step toward reducing the complexity of information spaces is to reduce the sensing requirements. Is it necessary for a robot to build an exact map of its environment? Is knowing its exact position important for solving basic navigation tasks? What about pursuit-evasion tasks? It has been shown in numerous robotics works that simple robots with little or no sensing can accomplish complicated tasks. Much of the early work in this direction was in the context of object manipulation (Goldberg 1993; Mason 2001). Other work includes bug algorithms for navigation (Lumelsky & Stepanov 1987; Kamon, Rivlin, & Rimon 1999), gap navigation trees for optimal navigation and on-line algorithms for pursuit-evasion (Kameda, Yamashita, & Suzuki 2003; Tovar, Guilamo, & LaValle 2004). The fo-

*This work is partially supported by DARPA HR0011-05-1-0008 and ONR N000014-02-1-0488 grants.

cus on minimal sensors and simple robots also appears in (Vaughan *et al.* 2000; Brown *et al.* 2002); these works, however, do not address the theoretical problem of connecting minimal sensor requirements with the difficulty of a given robotic task.

The current paper continues even further along the lines of reducing sensor requirements for given tasks, leading to robots that are so simple and sensor-limited that it appears absurd to expect them to solve these tasks. It is assumed that the robot has a single sensor that indicates whether or not the robot is in contact with a wall. The robot is called a *Bitbot*.

In this paper several challenging online tasks will be assigned to Bitbots. A single Bitbot will be placed in an unknown environment and asked to learn the environment, to localize itself, and to locate any moving targets. At first it seems impossible to accomplish these tasks with such a simple robot. However, in this paper we carefully model the Bitbots and show how the actions and sensor measurements change the representation of their belief about the environment and the moving targets. We formally characterize these spaces of information and present the solutions to the online tasks.

This work provides an illustration of the advantages of reasoning in terms of information spaces, and in the design of minimalist robotic systems. Designing a robotic system so that the resulting information space is manageable is crucial to the success of planning algorithms.

Bitbot Model

We define a Bitbot as a robot which can choose among two types of movements in a polygonal environment. First, it can follow the walls in either direction. Second, when approaching a reflex vertex v (Figure 1), it can choose to go straight of the reflex vertex along the continuation of the edge, called the *cut*, and land on the opposite edge of the environment. The Bitbot is equipped with a contact sensor, which indicates whether or not there is a contact with the boundary of the environment. These movements and sensor capabilities allow a Bitbot to differentiate between the states where it is in the contact with the walls and the states where it is moving along the cuts. Moreover, this information can be further reduced for determining whether there is a contact with a reflex vertex, a non-reflex vertex, or the Bitbot has just landed on an edge after following a cut.

First, we give some preliminary definitions, followed by the formal model of a Bitbot. Although we present a purely abstract model, it closely corresponds to a real robot model developed by (Lamperski *et al.* 2005).

Preliminaries

Consider the set of all the environments E for a Bitbot, such that each $e \in E$ has a simple polygonal boundary in $\mathbb{R}^2$ with the set of vertices $V^e = \{v_i \mid i = 0, \ldots, n-1\}$. We call the set of all reflex vertices V_r^e.

Given a reflex vertex $v \in V_r^e$, consider an edge incident to this vertex and the interval $[v, v_c]$, which is the maximal extension of this edge inside e. This interval is called a *cut* at v, with v_c being the cut's endpoint (Figure 1).

For each reflex vertex there are two cuts, corresponding to the two incident edges at this vertex. An example of a polygon with the set of all of the cuts is shown on Figure 1. We make a general position assumption, that no cut can contain a reflex vertex. Call V_c^e the set of all the endpoints v_c of the cuts in e, and C^e the set of all the cuts in e. Call $Q^e = V^e \cup V_c^e$ the *states* of a Bitbot for the environment e, that is the set of all the points on the boundary of the environment between which the Bitbot can differentiate.

Formal Model

Formally, a Bitbot can be described as a collection of the following: the set of states where the Bitbot can be, but that is unknown to the Bitbot; the set of observations, or sensor readings that the Bitbot can obtain at any state; the actions it can perform and the information about the transition that might occur in the state following such actions. We define all of these formally.

- The *state space*, $X = \mathbb{R}^2 \times E$, where each valid state $x = (q, e), q \in Q^e, e \in E$ represents the Bitbot position q with respect to the environment e it is in.
- The *observation space* for the contact sensor, $Y_c = \{$**reflex**, **nonReflex**, **cutEnd**$\}$.
- The *action space*, $U_c = \{$**goRight**, **goLeft**, **goRightOf**, **goLeftOf**$\}$, which represents the actions to move right and left along the walls, or right and left along the walls followed by going of along the cut.
- The *contact sensor mapping*, $h_c : X \to Y_c$, defined as

$$h_c(q,e) = \begin{cases} \textbf{nonReflex}, \text{ if } q \in V^e \setminus V_r^e \text{ in } e; \\ \textbf{reflex}, \text{ if } q \in V_r^e \text{ in } e; \\ \textbf{cutEnd}, \text{ if } q \in V_c^e \text{ in } e. \end{cases}$$

- The *state transition function*, $f_c^e : Q^e \times U_c \to Q^e$, defined for each $e \in E$ as

$$f_c^e(q,u) = \begin{cases} v_i^+, & \text{if } q = v_i, u = \textbf{goLeft}; \\ v_i^+, & \text{if } q \in V_c^e, q \in [v_i, v_i^+], u = \textbf{goLeft}; \\ v_i^-, & \text{if } q = v_i, u = \textbf{goRight}; \\ v_i, & \text{if } q \in V_c^e, q \in [v_i, v_i^+], u = \textbf{goRight}; \\ v_c, & \text{if } v = f_c^e(q, \textbf{goRight}) \in V_r^e; \\ & [q, v_c] \supset [v, v_c] \in C^e, u = \textbf{goRightOf}; \\ v_c, & \text{if } v = f_c^e(q, \textbf{goLeft}) \in V_r^e; \\ & [q, v_c] \supset [v, v_c] \in C^e, u = \textbf{goLeftOf}, \end{cases}$$

where $v_{i-1}^+ = v_{(i+1) \bmod n}$ and $v_{i-1}^- = v_{(i-1) \bmod n}$.

Task 1: Learning the Environment

This section formulates the first task for a Bitbot. When placed in an unknown environment, what can a single Bitbot learn about the geometry of this environment and about its position with respect to this environment?

The sensing limitations of Bitbots are so strong that learning sufficient information is impossible without an additional capability. The Bitbot is given a pebble in this section in order to complete the task.

We first describe the data structure that holds the topological representation of the environment that the Bitbot can

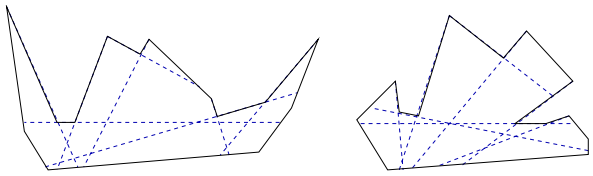

Figure 2: Some polygons having the cut diagram shown in Figure 1.

learn. Next we formulate the task as the search in the information space and give the solution. We also prove that a pebble is crucial for performing this task. A Bitbot with no pebble cannot learn this information about the environment.

Cut Diagram

In correspondence to the environment $e = (Q^e, C^e)$ consider a circle, s, with the set of points on the boundary, Q^e_D, and the set of chords, C^e_D.

Definition 1 *A collection $D^e = \{s, Q^e_D, C^e_D\}$ is called a cut diagram corresponding to the environment e iff*

- *There is a bijection $g_1 : Q^e \to Q^e_D$, such that the ordering of points in Q^e along the perimeter of e is preserved by g_1 in Q^e_D along the circle s.*
- *There is a bijection $g_2 : C^e \to C^e_D$, such that for every cut $[v, v_c] \in C^e$, $g_2(c) = [g_1(v), g_1(v_c)] \in C^e_D$.*

That is, a cut diagram represents all the cuts in the polygon drawn as chords on the circle between the corresponding vertices. An example of a cut diagram corresponding to a polygon is shown on Figure 1. The following property can be observed about the cut diagram.

Proposition 2 *A pair of chords in the cut diagram intersect iff the corresponding cuts intersect in the polygon.*

Proof: For all the proofs please refer to (Yershova *et al.* 2005) ∎

Problem Formulation

In this section we enhance a formal model of the Bitbot with the pebble and pose the problem of building the cut diagram of an environment by such a robot. This can be viewed as the search for the correct cut diagram in the space of all possible cut diagrams of all the environments. Therefore, the space of all the cut diagrams is the information space for this problem, and the task for the Bitbot can be considered as the search for the right cut diagram in this information space. The initial information state is an empty cut diagram, which represents the Bitbot's absence of knowledge about the number of vertices in the diagram or the cuts. Whenever the Bitbot discovers the first cut it can fill in a part of the cut diagram, reducing the uncertainty. This change in the information space is represented by the information state transition function. The final information state is the correct cut diagram of the environment. The goal for the Bitbot is to reach this state. It should be noted that a correct cut diagram still does not eliminate complete uncertainty about the geometry of the polygon. For example, Figure 2 shows some polygons having the cut diagram shown in Figure 1.

Now we define the problem of learning the environment.

- The *state space*, $X = \mathbb{R}^2 \times \{\mathbb{R}^2 \cup \emptyset\} \times E$, where each valid state $x = (q_r, q_p, e), q_r \in Q^e, q_p \in \{Q^e \cup \emptyset\}, e \in E$ represents the Bitbot position, pebble position and the environment they are in.
- The *observation space*, $Y = Y_c \times Y_p$, where $Y_p = \{$**seePebble**, **seeNoPebble**$\}$.
- The *action space*, $U = U_c \times U_p$, where $U_p = \{$**dropPebble**, **pickPebble**$\}$.
- The *contact sensor mapping*, h_c, defined as before.
- The *pebble sensor mapping*, $h_p : X \to Y_p$, defined as

$$h_p(q_r, q_p, e) = \begin{cases} \textbf{seePebble}, \text{ if } q_r = q_p \text{ in } e; \\ \textbf{seeNoPebble}, \text{ if } q_r \neq q_p \text{ in } e. \end{cases}$$

- The *state transition function*, $f : X \times U \to X$, defined as

$$f((q_r,q_p,e),u) = \begin{cases} (f^e_c(q_r,u), q_p, e), \text{ if } u \in U_c; \\ (q_r,q_r,e), \text{ if } q_p = \emptyset, u = \textbf{dropPebble}; \\ (q_r,\emptyset,e), \text{ if } q_p = q_r, u = \textbf{pickPebble}. \end{cases}$$

- The set of information states, $\eta_k = (\{y_i\}_{i=1}^k, \{u_i\}_{i=1}^{k-1})$. Each of the information states η_k represents the current subset D^e_k of the cut diagram D^e through the mapping $\eta(\eta_k) = D^e_k$.
- The initial information state, η_0, represents an empty cut diagram, which holds no information about cuts or vertices. That is $\eta(\eta_0) = D^e_0 = (s, \emptyset, \emptyset)$.
- The information state transition function represents the change in the current cut diagram after a new action has been performed and a new observation has been received. We do not specify it explicitly due to its size; however, the algorithm in the next section uses it implicitly.
- The goal is to find a sequence $\{u'_i\}_{i=1}^m, u_i \in U$, such that $\eta(\{y'_i\}_{i=1}^{m+1}, \{u'_i\}_{i=1}^m) = D^e$ for any possible $\{y'_i\}_{i=1}^{m+1}, y'_i \in Y$.

The Algorithm

To solve the problem the Bitbot proceeds in several stages (see Figure 3). First, it learns the number of edges in the environment. It achieves it by dropping a pebble and moving along the perimeter while counting the vertices until the pebble is reached again. Next, it systematically learns all the cuts in the environment, by executing the trajectory, which follows the perimeter just before and after the cut, for each of the cuts. Note, that the Bitbot builds the cut diagram by keeping track of its current position in it. Therefore, it is always localized with respect to the cut diagram.

So far the Bitbot has learned which cuts land at which edges. It can learn the cut diagram further and, by placing the pebble at the endpoint of each cut, learn the ordering of all the endpoints of all the cuts landed on the same edge.

Proposition 3 *A Bitbot with no pebble is not able to construct a cut diagram of the environment.*

CONSTRUCT_CUT_DIAGRAM
Given The set of actions U, the set of observations Y, the sensor mappings h_c and h_p, the state transition function f
Output The cut diagram D

```
1   dropPebble; i = 0
2   do goLeft
3       if y_c = reflex add v_i to V^e_rD
4       add v_i to V^e_D and i++
5   until y_p = seePebble
6   for each v ∈ V^e_rD
7       for each u ∈ {goLeft, goRight} ⊂ U_c
8           apply u index_in_V^e_D(v) times
9           if u = goLeft
10              then goLeftOf at v reaching v_c
11              else goRightOf at v reaching v_c
12          apply u and count index_in_Q^e_D(v_c)
13          until y_p = seePebble
14          initialize the cut [v, v_c] in C_D
```

Figure 3: The algorithm for constructing the cut diagram of an unknown environment

Task 2: Pursuit-Evasion

The second task for a Bitbot is more challenging. Consider arbitrarily moving targets in the environment. Can the Bitbot locate them? Having no vision sensors, can it learn where the targets may be?

Of course, without any additional sensor for detecting the targets the Bitbot can never "detect" them in the usual sense. However, we consider a slightly modified pursuit-evasion problem, in which the evaders are moving unpredictably but are willing to be found (for example, lost people in the building on fire). When they (the evaders) see the Bitbot, they are considered to be detected. Alternatively, an additional sensor for detecting the presence (though not the location) of targets can be given to a Bitbot.

In this section we show how a Bitbot can find all of the moving targets in the environment. We also prove the completeness of the algorithm.

Visibility Information of the Cut Diagram

The visibility region which can be seen from the Bitbot position defines the set of points, from which the Bitbot itself is visible. Therefore, if the goal of the Bitbot is to make itself visible by the evaders in the environment, the standard visibility concepts can be adapted to this problem.

First, we define the *visibility region*, $V(x)$, which is the set of all points in e that are visible from x. Call $e \setminus V(x)$ the *shadow region*, which is the set of all the points in e that are not visible from x. There may be several connected components in the shadow region. Each of them can be associated with some reflex vertex in the environment. We call each of them *a shadow component*. The common boundaries of the visibility region and the shadow region are called *gaps* in the visibility. That is, each gap corresponds to a line segment in e, which touches ∂e in two places, and if extended to a line, contains x.

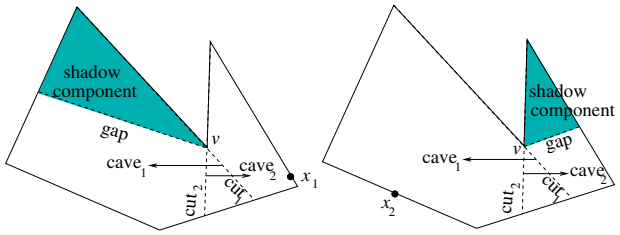

Figure 4: For a reflex vertex v the two cuts and corresponding caves are shown. For different positions x_1 and x_2 on the boundary, the gap associated with v and corresponding shadow component may be inside only one of the caves.

Traditionally, the pursuit-evasion problem is solved by keeping track of all the shadow components in the environment, since they are the only places where the evaders can hide (Guibas *et al.* 1999). This is done by labeling each gap as *contaminated* (an evader may be hiding behind the gap), or *cleared* (there is no evader behind the gap).

It was shown in the pursuit-evasion literature that the labeling of the gaps change only when certain critical events happen, i.e. when the robot crosses certain lines in the environment. These critical gap events are: 1. Crossing the cut. In this event either a new gap appears or an old gap disappears. 2. Crossing the bitangent between two reflex vertices. In this case either two gaps merge, or one gap splits into two. The gap labels change when the critical events happen. When the gap emerges, it is labeled as clear, since this portion of the environment was just visible. When the gap splits into several gaps, all of them inherit the label of the original gap. When the gaps merge into one, the new gap is labeled as contaminated, if one of the children gaps was contaminated.

A Bitbot does not have a sensor for tracking the gaps, however it can detect the cuts. This section shows how cuts can be used to label the gaps. With each cut in the environment we associate corresponding cave (Figure 1) and the gap that might or might not lie in the cave (Figure 4). Therefore, each gap can be associated with two cuts in such a way that the two cuts labels fully define the gap label.

We call a cut associated with the reflex vertex v *clear*, if the shadow region of v inside the cave of the cut is either clear or there is no shadow region associated with v in the cave. We call the cut *contaminated* if the shadow region of v inside the cave is contaminated. Then, one of the two cuts of each reflex vertex has a label clear, since the gap can only belong to one cave. Another one has the same label as the gap. The critical cut events are the same as the critical gap events: crossing the cut and crossing the bitangent. The difficulty is that the Bitbot does not know where the bitangents are. Instead, it has to learn the regions in the cut diagram, where the recontaminations may happen. Entering such regions leads to merges of the corresponding shadow components and therefore possible relabeling of the cuts.

To obtain recontamination regions in the cut diagram, recall that the cut diagram preserves two-intersections of the cuts (Proposition 2). Therefore, the positions of the bitan-

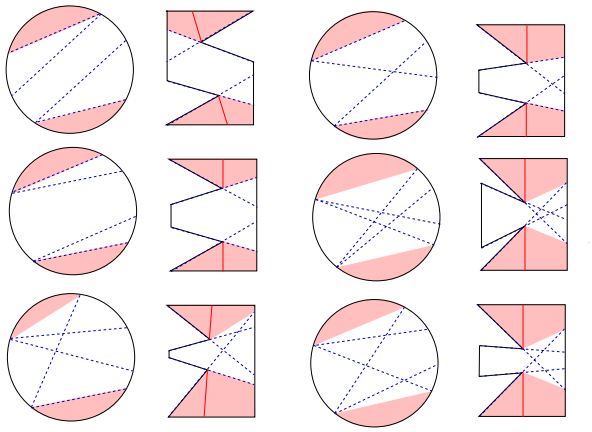

Figure 5: The configurations of the cuts of two reflex vertices which may result in a bitangent or recontamination region. The shaded regions are the places where recontamination may happen (where the bitangent may be crossed).

gents are inferred from the information of how the corresponding chords intersect in the cut diagram.

Proposition 4 *The only mutual configurations of the cuts in the cut diagram which may lead to recontaminations are the six cases shown on Figure 5.*

The recontamination regions have to be reduced even further when other cuts are present in the cut diagram. It should be done in such a way, that for any cut crossing the recontamination region the corresponding environment can be constructed, such that recontamination happens between the corresponding shadow components in the environment. We do not specify the details due to the space limit.

Unfortunately, the cut diagram does not preserve the three-intersections. That is, if two intersecting chords are intersected by a third chord, the order in which they are crossed in the cut diagram may be different from that in the polygon. This may affect the labeling of the cuts which the Bitbot crosses when going of some other cut in the environment. What if merging of the shadow components is not considered in the correct order? Does it result in the wrong labeling of the cuts? Fortunately, merging of the shadow components has a transitive property in the cut diagram, and therefore, any order, including the one that is formed in the cut diagram, results in the same labeling of the cuts.

Proposition 5 *Consider three reflex vertices in the cut diagram forming three corresponding recontamination regions. Any chord in the cut diagram intersecting any two of these regions will always intersect the third one.*

Problem Formulation

Given the cut diagram D of the environment e that the Bitbot has already learned, the task now it to locate all the evaders in the environment, which is equivalent to clearing all the cuts. Next we formulate the pursuit-evasion task.

- The *state space*, $X = \mathbb{R}^2$, where each valid state $x \in Q_D$ represents the Bitbot position in the cut diagram D.

PURSUIT_EVASION(D, η, x)
Input The cut diagram D, initial information state $\eta = \eta_0$, initial state x
Output The sequence $\{u'_i\}_{i=1}^m, u_i \in U$

```
1  add (η, x) to S
2  for all (η, x) from S
3      for all inputs u ∈ U
4          compute next state x_1 = f(x, u)
5          compute information state η_1 = f_I(η, x, u)
6          if (η_final = η_1) return {u'_i}_{i=1}^m
7          add (η_1, x_1) to S
8      remove (η, x) from S
9  return " NO SOLUTION EXISTS "
```

Figure 6: The algorithm for pursuit-evasion

- The *observation space*, $Y = X$.
- The *action space*, $U = U_c$.
- The *sensor mapping*, $h : X \to X$, defined as trivial mapping.
- The *state transition function*, $f : X \times U \to X$, such that $f(x, u) = g_1(f_c((g_1^{-1}(x), e(D)), u))$.
- The set of information states, $\eta_k = (\{y_i\}_{i=1}^k, \{u_i\}_{i=1}^{k-1})$. Each of the information states η_k represents the set of labellings $(l_1, \ldots, l_m)$ of m cuts, through the mapping $\eta(\eta_k) = (l_1, \ldots, l_m)$.
- The initial information state, η_0, represents the labellings of all the cuts that contain gaps as contaminated, and all others as clear.
- The information state transition function, which corresponds to the relabeling of the cuts while transferring from state to state, $f_{\mathcal{I}}((l_1, \ldots, l_m), x, u) = (l'_1, \ldots, l'_m)$, in the following way. Consider a path from x to $f(x, u)$ in the cut diagram. Take all the labels $\{l_i\}_{i \in I}$ of the cuts crossed by this path. Set $l'_i = l_i, \forall i \notin I$, and $l'_i = clear, \forall i \in I$. Reset the labels to *contaminated* for those cuts, whose recontamination regions with contaminated cuts were entered.
- The goal is to find a sequence $\{u'_i\}_{i=1}^m, u_i \in U$, such that $\eta(\{y'_i\}_{i=1}^{m+1}, \{u'_i\}_{i=1}^m) = (clear, \ldots, clear)$.

The Algorithm

The algorithm performs a breadth-first search on the graph of all the information states. From a starting information state all possible inputs are tried, relabeling is performed, resulting information state is computed and checked with the goal state. The same procedure is iterated over all the resulting states.

Proposition 6 *The algorithm presented above is Bitbot-complete, that is a Bitbot will find the solution if one exists.*

Implementation and Simulation Results

We have implemented both algorithms using Python. We performed experiments with three different environments.

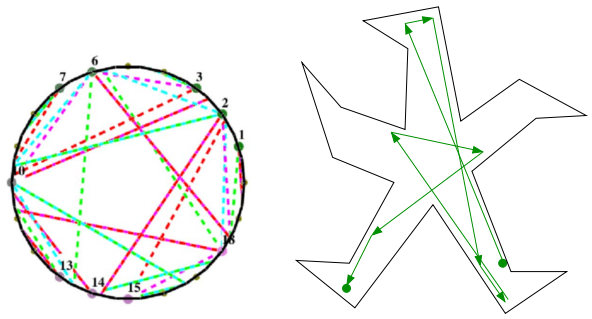

Figure 7: For the given environment, the cut diagram with all the cuts and recontamination regions, generated by our program, is shown on the left. The solution path for the pursuit-evasion algorithm is shown on the right.

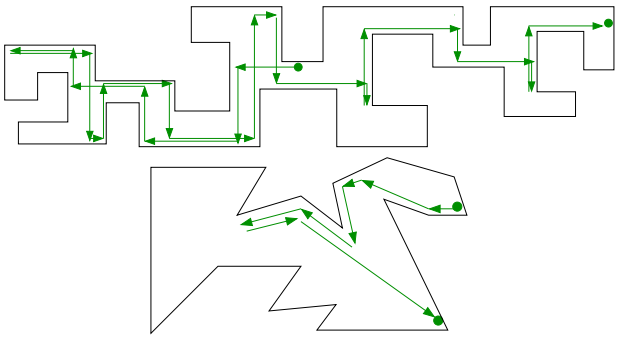

Figure 8: Computed solution paths for catching evaders in the given environments.

The first experiment is shown on Figure 7. We show the cut diagram learned by a Bitbot with all the cuts and recontamination regions. Using only the cut diagram and the algorithm for pursuit-evasion, the Bitbot generates the plan for finding all the evaders in the environment. The produced solution path is shown. In the experiments shown on Figure 8 the pursuit-evasion algorithm was used to produce the clearing paths for the other two environments.

Conclusions

In this paper we introduced very simple robots called Bitbots. The sensor and control limitations of Bitbots allow careful modeling of the information spaces arising when presenting online tasks to these robots. We have formally defined the tasks of map building, localization and pursuit-evasion as planning problems in corresponding information spaces and presented solutions and simulation results.

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

References

Brown, H. B.; Weghe, M. V.; Bererton, C.; and Khosla, P. 2002. Millibot trains for enhanced mobility. *IEEE/ASME Transactions on Mechatronics* 7(4):452 – 461.

Choset, H., and Nagatani, K. 2001. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Int. Conf. Robot. & Autom.* 17(2):125–137.

Gerkey, B. P.; Thrun, S.; and Gordon, G. 2004. Visibility-based pursuit-evasion with limited field of view. In *Proc. Am. Assoc. Artif. Intell.*

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

Guibas, L. J.; Latombe, J.-C.; LaValle, S. M.; Lin, D.; and Motwani, R. 1999. Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications* 9(5):471–494.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Kameda, T.; Yamashita, M.; and Suzuki, I. 2003. On-line polygon search by a six-state boundary 1-searcher. Technical Report CMPT-TR 2003-07, School of Computing Science, SFU.

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

Lamperski, A. G.; Loh, O. Y.; Kutscher, B. L.; and Cowan, N. J. 2005. Dynamical wall-following for a wheeled robot using a passive tactile sensor. In *IEEE Int. Conf. Robot. & Autom.*

Lumelsky, V. J., and Stepanov, A. A. 1987. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica* 2:403–430.

Mason, M. T. 2001. *Mechanics of Robotic Manipulation*. Cambridge, MA: MIT Press.

Montemerlo, M.; Thrun, S.; Koller, D.; and Wegbreit, B. 2002. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI National Conference On Artificial Intelligence*.

Thrun, S.; Fox, D.; Burgard, W.; and Dellaert, F. 2001. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence Journal*.

Tovar, B.; Guilamo, L.; and LaValle, S. M. 2004. Gap navigation trees: Minimal representation for visibility-based tasks. In *Proc. Workshop on the Algorithmic Foundations of Robotics*.

Vaughan, R.; Stoey, K.; Sukhatme, G. S.; and Mataric, M. J. 2000. Blazing a trail: insect-inspired resource transportation by a robot team. In *Proc. 5th International Symposium on Distributed Autonomous Robotic Systems*, 111–120.

Yershova, A.; Tovar, B.; Ghrist, R.; and LaValle, S. M. 2005. Bitbots: Simple robots solving complex tasks. Technical Report, Computer Science Dept., University of Illinois (http://msl.cs.uiuc.edu/∼yershova/ai2005/paper.pdf).