[4] V. D. Gligor and S. H. Shattuck, "On deadlock detection in distributed systems," *IEEE Trans. Software Eng.*, vol. SE-6, no. 5, Sept. 1980.
[5] E. M. Gold, "Deadlock prediction: Easy and difficult cases," *SIAM J. Comput.*, vol.7, no. 3, Aug. 1978.
[6] J. N. Gray, "Notes on data base operating systems," in *Operating Systems—An Advanced Course*, vol. 60, Bayer, Graham, and Seegmuller, Eds. New York: Springer-Verlag, 1978.
[7] A. N. Habermann, "Prevention of system deadlock," *Commun. ACM*, vol. 12, p. 373, 1969.
[8] R. C. Jolt, "Comments on prevention of system deadlock," *Commun. ACM*, vol. 14, p. 36, 1971.
[9] J. H. Howard, "Mixed solutions for the deadlock problem," *Commun. ACM*, vol. 16, p. 427, 1973.
[10] S. Joshi, R. A. Wysk and A. Jones, "A scaleable architecture for CIM shop floor control," in *Proc. CIMCON'90* (National Institute of Standards and Technology, May 22–24, 1990), pp. 21–33.
[11] D. A. Menasce and R. R. Muntz, "Locking and deadlock detection in distributed data bases," *IEEE Trans. Software Eng.*, vol. SE-5, no. 3, May 1979.
[12] S. Tsutsui, and Y. Fujimoto, "Deadlock prevention in process computer systems," *Comput. J.*, vol. 30, no. 1, 1987.

Robotic Exploration as Graph Construction

Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes

Abstract—**We address the problem of robotic exploration of a graph-like world, where no distance or orientation metric is assumed of the world. The robot is assumed to be able to autonomously traverse graph edges, recognize when it has reached a vertex, and enumerate edges incident upon the current vertex relative to the edge via which it entered the current vertex. The robot cannot measure distances, and it does not have a compass. We demonstrate that this exploration problem is unsolvable in general without markers, and, to solve it, we equip the robot with one or more distinct markers that can be put down or picked up at will and that can be recognized by the robot if they are at the same vertex as the robot. We develop and prove correct an exploration algorithm, we show its performance on several example worlds, and we discuss heuristics for improving its performance.**

I. Introduction

Robotic exploration and map-based navigation using dead reckoning is difficult to accomplish over large spatial scales without reference to external features of the world due to the accumulation of error in robot position and orientation. To address this fundamental issue, a number of approaches, e.g., [4], have been suggested in order to more accurately track the motion of a robot in relation to features of its environment so as to reduce positional errors.

Manuscript received April 28, 1989; revised February 20, 1991. This work was supported by the National Sciences and Engineering Research Council of Canada and by the Government of Ontario the Information Technology Research Center. Portions of this work were presented at the SPIE Symposium on Advances in Intelligent Robotics Systems, Philadelphia, PA, November 1989, and at Vision Interface, London, Ontario, June 1989.

G. Dudek is with the McGill Research Center for Intelligent Machines, McGill University, Montreal, Quebec, Canada H3A 2A7.

M. Jenkin and E. Milios are with the Department of Computer Science, York University, Toronto, Canada M3J 1P3.

D. Wilkes is with the Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4.

IEEE Log Number 9102768.

Motivated by the need for spatial representations other than ones based on metric information, a four-level spatial semantic hierarchy is proposed in [15]. The four levels, starting with the lowest, are the sensorimotor (robot sensations and primitive actions), procedural (robot actions to accomplish place-finding and route-following tasks), topological (places and paths and their topological relations), and metric (places and paths and their metric relations). In this paper, we provide precise definitions of what correspond to the sensorimotor, procedural, and topological levels associated with learning and navigation in a graph-like world, assuming no metric information is either sensed or stored.

The problem addressed in this work is: Given an unknown environment modeled as a graph, formulate an exploration strategy for the robot, so that, after carrying out the actions specified by the strategy, the robot will have formed a representation of its environment sufficient for solving navigation tasks with the use of sensors, i.e., a map. We begin by defining the world the robot explores, the robot itself, and the actions and sensations with which it is equipped. To solve the problem of determining when the robot has returned to a previously visited location (the "am I there yet" problem) during map building, we use portable markers as part of the exploration strategy. We demonstrate that this is the only feasible approach given the lack of precise metric information and distinct landmark features.

Past work in this area involves metric representations of space, including metric graphs [4], [13], [16], [19], [21], and probabilistic approaches, which typically associate probability distributions with spatial coordinates, [3], [9], [10], [17], [18], [20]. Graph theoreticians have examined the complexity required of a machine to traverse (without building a model of) a two- or three-dimensional space with obstacles [1], [2]. A major difference between random walks on graphs and our approach is that we visit vertices of a graph according to deterministic strategies as opposed to stochastic. In [14] and [15] a topological model of the world is used. A rehearsal procedure is used to distinguish two places with identical signatures by exploring some of the surrounding area. Our work can be viewed as introducing a different rehearsal procedure that includes portable markers. Davis [5] has tried to reduce problems at the topological level to problems involving only the geometric level, which are then solved by trigonometric means applied to fuzzy intervals. Permanent markers have been used in deterministic strategies for graph traversal [11], where the goal is to visit all vertices of an unknown graph. The problem of exploring an unknown directed graph with a robot that can recognize edges and nodes, if it sees them again, has been addressed by Deng and Papadimitriou [6].

This short paper is structured as follows. Section II defines our model of the world and the sensorimotor capabilities of the robot. Section III describes an exploration algorithm using k markers and gives a correctness proof and complexity bounds. Section IV presents experimental results on typical graphs and on a complex three-dimensional maze.

II. Definitions

In this section, we describe the environment in which our robot explorer exists as well as the robot's actions, perception, and goals. It is important to remember that this representation is minimal in nature. In reality, much more information is likely to be available at a given location than we assume here. A sensing system that can enumerate the exits from a given location might also give approximate relative orientations, and the addition of a compass to the robot would make these orientations absolute.

The robot's "purpose in life" is to use its ability to act and to perceive in the graph domain in order to build a graph embedding that is isomorphic [12] to the finite world it has been assigned to explore. The robot's inputs are its sensations, and it can interact with the world only through its actions.

A. The World

The world is defined as an embedding of an undirected graph G:

$$G = (V, E) \tag{1}$$

with a set of vertices V and a set of edges E. The vertices are denoted by

$$V = \{v_1, \cdots, v_N\}. \tag{2}$$

We will restrict the world model to graphs G that contain no cycles of length ≤ 2, i.e., the graph contains no degenerate or redundant paths. This restriction prohibits the world from having multiple edges between two vertices or an edge incident twice at the same vertex.

The definition of an edge is extended slightly to allow for the explicit specification of the order of edges incident upon each vertex of the graph embedding. This ordering is obtained by enumerating the edges in a systematic (e.g., clockwise) manner from some standard starting direction. An edge $E_{i,j}$ incident upon v_i and v_j is assigned labels n and m, one for each of v_i and v_j, respectively. n represents the ordering of the edge $E_{i,j}$ with respect to the consistent enumeration of edges at v_i; m represents the ordering of the edge $E_{i,j}$ with respect to the consistent enumeration of the edges at v_j. The labels m and n can be considered as general directions, e.g., from vertex v_i the nth exit takes edge $E_{i,j}$ to vertex v_j.

B. Movement and Action

The robot can move from one vertex to another by traversing an edge (a *move*), it can pick up a marker that is located at the current vertex, and it can put down a marker it holds at the current vertex (a *marker operation*). The robot in general has K markers at its disposal.

Assume the robot is at a single vertex v_i, having entered the vertex through edge $E_{i,l}$. In a single move, it leaves vertex v_i for vertex v_j by traversing the edge $E_{i,j}$, which is r edges after $E_{i,l}$ according to the edge order at vertex v_i (see Fig. 1). This is given by the transition function

$$\delta(v_i, E_{i,l}, r) = v_j. \tag{3}$$

We assume the following property about the transition function: if $\delta(v_i, E_{i,l}, r) = v_j$ and $\delta(v_j, E_{i,j}, s) = v_k$, then $\delta(v_j, E_{j,k}, -s) = v_i$. This implies that a sequence of moves is invertible, and can be retraced. We also assume that there does not exist a $t \neq -s$ such that $\delta(v_j, E_{j,k}, t) = v_i$.

A single move is thus specified by the order r of the edge along which the robot exits the current vertex, where r is defined with respect to the edge along which the robot entered such vertex. Note that in the special case of a planar embedding of a graph, enumeration of edges in a clockwise fashion satisfies the above assumption.

A marker operation is fully specified by indicating for each of the K markers whether it is being picked up, put down, or not operated upon. This is specified by a K-tuple $\Omega^K = (op_1, op_2, \cdots, op_K)$, where the element op_k has a value from the set $\{pickup, put\text{-}down, null\}$, according to the operation performed on marker k.

A simple action a is defined as a marker operation accompanied by a move; therefore, $a = (b, \delta)$, where $b \in \Omega^K$. The robot performs some action on the markers in the current vertex and then moves to a new location. A path $A \in a^+$ is a nonempty sequence of actions.

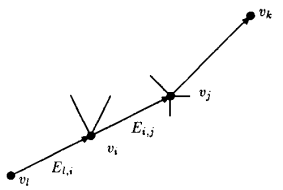

Fig. 1. Transition function. Assuming that the robot started at v_l, then the transition function δ identifies exit r from vertex v_i as being an edge to v_j.

C. Perception

The robot's perception is of two kinds: marker-related and edge-related perception.

Marker-Related Perception: Assume that the robot is at vertex v_i, having arrived via edge $E_{i,j}$. The marker-related perception of the robot is a K-tuple $B_s = (bs_1, bs_2, \cdots, bs_K)$, where bs_k has a value from the set $\{present, not\text{-}present\}$, according to whether marker k is present at vertex v_i.

Edge-Related Perception: The robot can determine the relative positions of edges incident on the vertex v_i in a consistent manner, e.g., by a clockwise enumeration starting with $E_{i,j}$. As a result, it can assign an integer label to each edge incident on v_i, representing the order of that edge with respect to the edge enumeration at v_i. The label 0 is assigned arbitrarily to the edge $E_{i,j}$ through which the robot entered vertex v_i. The ordering is local because it depends on the edge $E_{i,j}$. Entering the same vertex from two different edges will lead to two local orderings, one of which is a permutation of the other. Note that if the graph is planar and a spatially consistent (e.g., clockwise) enumeration of edges is used, then two permutations will be simple circular translations of each other. But, this will not hold in general, and in this work we only assume that the edges can be ordered consistently.

The sensory information that the robot acquires while at vertex v_i is the pair consisting of the marker-related perception at that vertex and the order of edges incident on that vertex, with respect to the edge along which the robot entered the vertex. If the robot visits the same vertex twice, it must relate the two different local orderings produced and unify them into a single global ordering, for example, by finding the label of the zeroth edge of the second ordering with respect to the first ordering. Determining when the same vertex has been visited twice and generating a global ordering for each vertex is part of the task of the exploration algorithm.

D. The Zero-Marker Case

Can an autonomous robot (as described above) explore an arbitrary finite environment without any markers? The answer is no. As an example, regular graphs of the same degree (i.e., graphs in which each vertex has a number of incident edges equal to the degree) are indistinguishable from each other without markers because each vertex appears identical to every other vertex.

III. The Exploration Algorithm

This section describes an algorithm for exploring a given graph using k markers. The algorithm will be shown to have polynomial running time in terms of the size of the graph. The basis of the algorithm is the maintenance of an explored graph. As new vertices are encountered, they are added to the explored graph, which is a subgraph of the full graph, and their outgoing edges are added to the

set of edges that lead to unknown places and therefore must be explored.

More formally, the algorithm maintains an explored subgraph S and a set of unexplored edges U, which emanate from vertices of the explored subgraph. A step of the algorithm consists of selecting a set E of k unexplored edges from U and "validating" the vertex v_2 at the unexplored end of each edge $e = (v_1, v_2)$ in the set E. Validating a vertex v_2 means making sure that it is not identical to any other vertex in the explored subgraph. This is carried out by placing one of the k markers at v_2 and visiting all vertices of the known subgraph S along edges of S, looking for the marker (and each of the other $k - 1$ markers dropped at this step). Note that the other vertex v_1 incident upon e is already in the subgraph S.

If the marker is found at vertex v_i of the explored subgraph S, then vertex v_2 (where the marker was dropped) is identical to the already known v_i (where the marker was found). In this case, edge $e = (v_1, v_2)$ must be assigned an index with respect to the edge ordering of vertex v_2. To determine this, the robot drops the marker at v_1 and goes back to v_2 along the shortest path in the explored graph S. At v_2, it tries going out of the vertex along each of its incident edges. One of them will take the robot back to v_1, which the robot will immediately recognize by the existence of the marker. Note that the index of e with respect to the edge ordering of v_1 is known by construction. Edge e is then added to the subgraph S and removed from U.

If the marker is not found at one of the vertices of S, then vertex v_2 is not in the subgraph S and therefore must be added to it. The unexplored edge e is also added to S, which has now been augmented by one edge and one vertex. Adding the vertex v_2 to the subgraph causes all edges incident upon it to be assigned an index with respect to the edge e by which the robot entered the vertex (edge e is assigned index 0) and the new edges are added to the set of unexplored edges U. Note that no other edge of the new vertex v_2 has been previously added to the subgraph because otherwise v_2 would have already been in the explored subgraph. This index assignment establishes the edge ordering local to v_2.

The algorithm terminates when the set of unexplored edges U is empty. A version of the algorithm incorporating the above characteristics is shown in the appendix. A formal proof of the correctness of the algorithm is presented in [8].

A different way of using the available k markers is to employ two distinct markers in the exploration of a single unexplored edge $e = (v_1, v_2)$. Then we can combine the validation and ordering steps by placing the markers at v_1 and v_2. If v_2 is found in S, then ordering of e with respect to v_2 is accomplished by going out of v_2 along each of its incident edges, without having to drop the marker at v_1 and to return to v_2 along the shortest path in S. This variation resulted in poorer performance on our test cases, with asymptotic worst case complexity that differs only by a constant factor. There is a tradeoff between easier vertex validation, with the modified algorithm, and fewer edges added per marker drop.

Complexity of the Algorithm

Certain steps of the algorithm are executed *mechanically* (edge traversals or marker pickup or drop) in the world G, while others are executed only *electronically* as the robot reasons about the model S (i.e., performs operations on the data structures maintained by the algorithm). The time constants for some simple operation, such as following an edge in a graph, may differ between the two cases by many orders of magnitude. Consequently, algorithms of higher asymptotic complexity may be tolerable for the electronic operations but not for the mechanical operations. For example, we may run a shortest-path algorithm electronically many times in order to save a few mechanical steps for the robot.

1) Mechanical Complexity: Let m be the number of edges and n the number of vertices. We will count the steps that may be taken by the robot while executing the algorithm in the worst case, using simplifying bounds on some subexpressions. We assume that k is small enough that the algorithm does not frequently exhaust the set U of unknown edges at the step in which it chooses k elements from U. If this is *not* the case, then the advantage of the extra markers is reduced, since the excess markers are not used and hence provide absolutely no advantage. By counting the number of mechanical steps resulting from each call to a relevant routine as shown in Table I, we obtain a bound on the number of mechanical steps of

$$\left(5 + \frac{2}{k}\right)mn - 3n^2 + \text{lower order terms} \tag{4}$$

where n_s is the number of nodes in the explored subgraph, and $d_{\max}$ is the maximum degree of a node in the world, both bounded by n. A similar analysis of the worst case asymptotic complexity of the modified algorithm, which places markers at each end of each unknown edge, gives a similar expression bounding the number of mechanical steps:

$$\left(3 + \frac{4}{k}\right)mn - n^2 + \text{lower order terms}. \tag{5}$$

2) Electronic Complexity: The asymptotic electronic complexity of the algorithm is $O(n^2m + m^2 \log m)$ if a shortest path algorithm is used in the routine *search*( ), or $O(m^2 \log m)$ otherwise. For planar graphs, this becomes $O(n^3)$ and $O(n^2 \log n)$, respectively. If the robot is aware that it has at least n markers, then some simplification is afforded because the robot need not pick up any markers that it drops.

IV. Experiments

Experiments were carried out by running the algorithm on various input graphs. These graphs were generated manually in order to exemplify performance characteristics of the technique and confirm the theoretical predictions. Equation (4) provides a theoretical upper limit on the number of robot steps required for the graph exploration. In practice, however, the actual number of steps for many operations may be substantially lower. In order to informally evaluate some of these issues, the behavior of a robot using this exploration strategy was simulated on a variety of input graphs. The actual number of mechanical steps taken varied between about 25 and 65% of the theoretical complexity bound.

An interesting example is provided by Daedalus' labyrinth, as described in [7]. We reproduce this maze in Fig. 2. Note that this maze is three dimensional with the entrance and exit located at the top level of the maze (Level 1). In order to test the ability of a robot equipped with a single marker, this maze was first reduced to a graph-based world model, with vertices in the graph representing intersections of paths or features (such as corners in the labyrinth). Two additional vertices were added for the entrance and exit rooms. The robot was then placed in the entrance room and allowed to start exploring. The robot knows which way is vertical, i.e., it has a sense of gravity but no sense of global orientation. The edge ordering follows the clockwise order for the horizontal edges, followed by the vertical up and vertical down edges, if any exist at the current vertex. It should be noted that this graph, as all graphs discussed in this paper, is undirected because vertical edges can be traversed both ways.

Internally the robot has no knowledge of the physical position of locations in the environment. It can only measure the exits from a

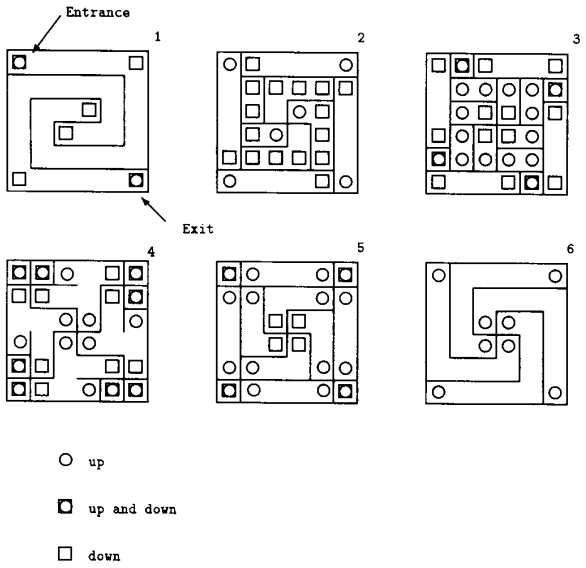

Fig. 2. Daedalus' labyrinth. This maze is constructed within a $6 \times 6 \times 6$ cube. The entrance and exit are on the top level (Level 1).

TABLE I
NUMBER OF MECHANICAL STEPS ASSOCIATED WITH THE ALGORITHM LISTED IN THE APPENDIX

	# of mechanical steps per call	# of calls in the algorithm
pickup(marker$_i$)	1	k
▷ marker distribution phase		
walk($\phi(v_{\text{current}})$, $\phi(v_1)$)	$\leq n_s - 1$	m
followEdge($\phi(e_i)$) ▷ *to* $\phi(v_2)$	1	m
drop(marker$_i$)	1	m
followEdge($\phi(e_i)$) ▷ *to* $\phi(v_1)$	1	m
search(·)		
breadth-first traversal	$\leq 2(n_s - 1)$	$\left\lceil \frac{m}{k} \right\rceil$
pickup(marker$_i$)	1	m (∗)
▷ validation for found marker		
walk($\phi(v_{\text{current}})$, $\phi(v_1)$)	$\leq n_s - 1$	$m - n + 1$
drop(marker$_i$)	1	$m - n + 1$
walk($\phi(v_1)$, $\phi(v_2)$)	$\leq n_s - 1$	$m - n + 1$
followEdge($\phi(f)$) ▷ *to* v_{unknown}	1	$(m - n + 1)d_{\max}$
pickup(marker$_i$)	1	$m - n + 1$
followEdge($\phi(f)$) ▷ *to* v_2	1	$(m - n + 1)d_{\max}$
▷ marker not found, new vertex		
walk($\phi(v_{\text{current}})$, $\phi(v_1)$)	$\leq n_s - 1$	$n - 1$
followEdge($\phi(e_i)$) ▷ *to* $\phi(v_2)$	1	$n - 1$
pickup(marker$_i$)	1	$n - 1$
pickup(marker$_j$)	1	included in (∗) above

node and the presence (or absence) of a marker. For display purposes only, we have attached to each node a physical location so that the resulting explored labyrinth can be compared with the original, shown in Fig. 2.

Two intermediate results with eight markers (after 1200 and 2400 mechanical steps) are shown in Fig. 3. The robot's perception of the maze after completing the exploration is identical to the original. Note that in Fig. 2 lines represent walls, while in Fig. 3 the lines represent valid movement paths for the robot.

The Advantages of Multiple Markers

As we have already mentioned, the use of multiple markers can improve the performance of the robot on a given environment. To confirm our previous observations on the effect of the number of markers on the robot's performance, we have plotted the total number of steps required to fully explore the maze with a number of markers ranging from 1 to 14, as shown in Fig. 4. The dots and crosses correspond to two different starting points in the maze.

It can be seen that adding more markers does not result in a significant decrease in the total number of steps after a certain number of markers. We conjecture that this is due to the fact that the number of unexplored edges at any one time are generally few, and that it may be the case that there are actually markers that cannot be used at a given step of the algorithm. Even if all the markers can be used, it may be the case that the robot must traverse much of the explored graph just to deposit the markers, and that the

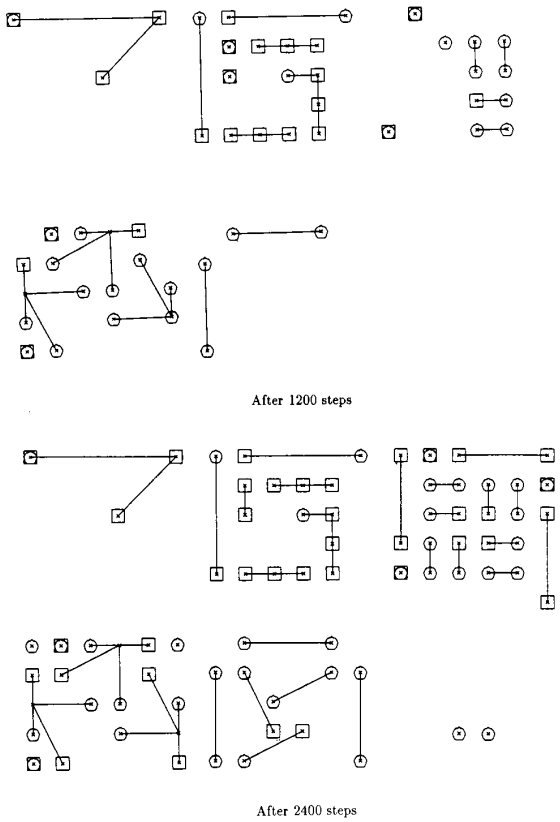

Fig. 3. Partial exploration results with eight markers.

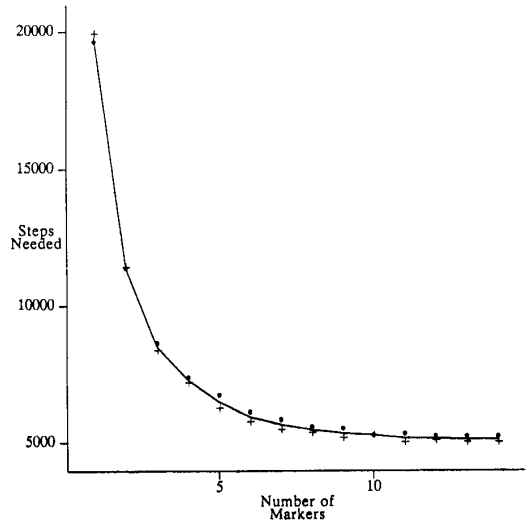

Fig. 4. Total number of steps as a function of the number of markers.

robot ends up taking a large number of steps when it would be more useful to simply explore the graph with the markers already dropped. Developing heuristics for improving the performance of the algorithm with multiple markers is an interesting area for future research.

V. Discussion

In this work, we proposed strategies that a mobile robot can use in exploring unknown graph-like environments. Contrary to most current approaches, our algorithms do not make use of distance metrics, but they use one or more markers that can be perceived by the robot and be put down or picked up at will. We show that one marker is sufficient to allow the robot to build a graph isomorphic to the environment in low-order polynomial time in both mechanical and computational complexity. The required perceptual abilities of the robot are well defined, and they amount to the ability to traverse an edge and to enumerate the edges incident upon the current vertex. The theoretically derived upper bounds on the complexity of the algorithm are confirmed by a computer simulation. By defining a minimal model for mobile robotic exploration, we point to interesting extensions for when the world is richer than our model.

The abstraction used here presents a particularly impoverished world. Only the number of exits from each location and one's own markers can actually be sensed. In practice, most real robotic systems can be expected to have richer perceptual inputs than these. The simple model used here, however, serves as a lowest common denominator for the capabilities of robotic systems. By showing that the map acquisition problem can be solved with acceptable complexity bounds under these circumstances, we demonstrate that such tasks are solvable within at least these bounds by more sophisticated systems. Furthermore, although more sophisticated perceptual mechanisms may be available, they are rarely completely dependable (not only must one account for sensor errors, but the sensed data is also domain dependent). Hence, even robots with powerful sensing systems may occasionally find themselves reduced to the level of the model described here. Finally, an abstraction of the robot exploration problem based on using more powerful but domain-dependent and error-prone sensors would make solid complexity bounds for the problem almost unobtainable.

It should be emphasized that the robotic exploration problem described here cannot be solved simply using a depth or breadth-first search. The identity of individual vertices of the graph cannot be established without first solving the mapping problem. Once the graph has been explored, it can then be searched efficiently by standard techniques (and the algorithm, in fact, does this within the portion of the graph that has already been mapped out).

Appendix

Statement of the Algorithm

The following is one version of the algorithm incorporating these assumptions. We denote the mapping from model S to world G by ϕ; ϕ of a vertex in S is the real-world vertex to which the robot's label corresponds. Definitions of the subroutines used by the algorithm are given following the statement of the top-level algorithm. In the top-level algorithm, mechanical steps are in italics, and electronic steps are in roman. Comments are preceded by the symbol ▷.

```
▷
▷ Initially the robot starts at some vertex v_initial with
▷ the markers on the floor
▷
for i from 1 to k
    pickup(marker_i)
S := ({v_initial}, { })
    U := {φ^-1(e) | e is incident with φ(v_initial)}
        ▷ U is indexed by the known vertex incident with
          each edge e
for each e in U
```

```
    index(e, v_initial) := consistent ordering of φ(e)
        with respect to an arbitrary edge
  end for
  v_current := v_initial
  loop
    exit when U = { }   ▷ i.e., no unexplored edge remains
    ▷
    ▷ Select a subset of the unexplored edges in U
    ▷ of up to k elements and call this E
    ▷
    choose(E)
    ▷
    ▷ For each new edge, move the robot there (v_1)
    ▷ Move along the new edge, drop the marker there (v_2)
    ▷ and return to the known graph (v_1).
    ▷
    for each e_i ∈ E, with known vertex v_1
      walk((v_current), φ(v_1))
      followEdge(φ(e_i)) ▷ to φ(v_2)
      drop(marker_i)
      followEdge(φ(e_i)) ▷ to φ(v_1)
    end for
    ▷
    ▷Search through the graph looking for the dropped
    ▷markers
    ▷
    search(S, markerFound, markerLocation)
      ▷ and set v_current
      ▷ markerFound is now a k-vector of Boolean flags
      ▷ markerLocation is a k-vector of vertex numbers
    for each i from k down to 1
      if markerFound_i then
        ▷
        ▷ Found a marker.
        ▷determine index of e_i
          with respect to its unknown end v_2:
        ▷
        walk(φ(v_current), φ(v_1))
        drop(marker_i)
        walk(φ(v_1), φ(v_2))
        for each edge f leaving v_2
            followEdge(φ(f)) ▷ to v_unknown
            if marker_i at v_unknown then
              index(e_i, v_2) := index(f, v_2)
              remove f from U
              pickup(marker_i)
              v_current := v_1
              exit for
            end if
            followEdge (φ(f)) ▷ back to v_2
          end for
        else
          ▷
          ▷ Didn't find the marker. This is a new vertex
          ▷
          walk(φ(v_current), φ(v_1))
          followEdge(φ(e_i)) ▷ to φ(v_2)
          pickup(marker_i)
          add v_2 to S
          add e_i to S
          index(e_i, v_2) := 0
          for each other edge f leaving φ(v_2)
            index(φ^-1(f), v_2) :=
                consistent ordering with respect to e_i
            add φ^-1(f) to U
          end for
          for each other marker_j at φ(v_2)
            pickup(marker_j)
            markerFound_j := true
            markerLocation_j := v_2
          end for
        end if
      end for
    end loop

  subroutines:
  choose(E)
    ▷ choose up to k edges e_1, e_2, ··· e_k from U
      such that the known
    ▷ incident vertex of e_1 is closest to v_current, and
    ▷ for i = 1, 2, ··· k − 1 we have that the known
      incident vertex of
    ▷ e_{i+1} is closest to the known incident vertex of e_i
    run shortestPath k times to find edges satisfying the
      above description.

  walk(v_from, v_to)
    run shortestPath to get shortest path (e_1, e_2, ··· e_k)
    from v_from to v_to, through S.
    for i from 1 to k
      followEdge(φ(e_i))
    end for
  search(S, markerFound, markerLocation)
    ▷ a breadth-first approximation
    ▷ to a traveling salesman problem solution seems
    ▷ appropriate
    ▷ since markers are likely to be close to current vertex
    ▷ in S
    ▷ Do traversal of S, stopping when k markers have
    ▷ been encountered
    ▷ or all vertices have been visited
    run Kruskal's algorithm to get min spanning tree of S
    for i from 1 to k set markerFound_i to false
    do breadth-first traversal,
      taking "short cuts" across non-tree
    edges to next vertex where possible.
      We consider two versions here:
      a) only take single-edge short cuts (check if current
         and next vertex in traversal are adjacent).
      b) run shortestPath to find shortest path from
         current to next vertex at each step.
    whenever a marker i is encountered,
      set markerFound_i to true, and
      set markerLocation_i to the vertex number in S, and
    execute pickup(marker_i).
  shortestPath(source)
    ▷ do a breadth-first labeling of vertices starting from
    ▷ vertex "source"
    ▷ where labels indicate previous vertex in path back
    ▷ to sink. This
    ▷ is inspired by the Ford-Fulkerson labeling algorithm.
    ▷ It suffices for finding shortest paths in an
    ▷ unweighted graph,
    ▷ taking O(n) time. Use Dijkstra's algorithm if there
    ▷ are weights on the edges (O(n^2)).
    label source
    loop
```

for each newly labeled vertex v
(i.e., from last loop pass)
label all vertices adjacent to v
end for
end loop

ACKNOWLEDGMENT

The authors would like to thank Prof. A. Borodin, Dr. I. Hartman, Dr. H. Everett, and Dr. B. Selman for useful discussions about graph traversal problems and the notion of isomorphism of graph embeddings. Prof. B. Kuipers, whose pioneering work in spatial reasoning provided the starting point for this paper, offered useful comments on a draft of the paper. Finally, we thank one of the anonymous reviewers for suggesting the possibility of using $2k$ markers to explore k edges, and both reviewers for their thoughtful comments on the original manuscript.

REFERENCES

[1] M. Blum and W. Sakoda, "On the capability of finite automata in 2 and 3 dimensional space," in *Proc. FOCS Conf.*, 1977, pp. 147-161.
[2] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa, "Two applications of complementation via inductive counting," Tech. Rep. 13179 (no. 58972), IBM Res. Div., 1987.
[3] R. Brooks, "A robust layered control system for a mobile robot," Tech. Rep. AIM-864, MIT AI Lab., 1985.
[4] J. Crowley, "Navigation for an intelligent mobile robot," *IEEE J. Robotics Automat.*, vol. RA-1, pp. 31-41, Mar. 1985.
[5] E. Davis, *Representing and Acquiring Geographic Knowledge*. London: Pitman and Morgan Kaufmann Publishers, 1986.
[6] X. Deng and C. Papadimitriou, "Exploring an unknown graph," in *Proc. Ann. Symp. Foundations Comput. Sci.*, 1990, pp. 335-361.
[7] A. Dewdney, "Old and new three-dimensional mazes," *Sci. Amer.*, Sept. 1988, pp. 136-139.
[8] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction," Tech. Rep. RBCV-TR-88-23, Univ. of Toronto, Res. in Biological and Computational Vision, 1988.
[9] H. Durrant-Whyte, "Uncertain geometry in robotics," *IEEE J. Robotics Automat.*, vol. 4, Feb. 1988.
[10] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 249-265, June 1987.
[11] S. Even, *Algorithmic Combinatorics*. New York: Macmillan, 1973.
[12] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams," *ACM Trans. Graphics*, vol. 4, 1984.
[13] S. Iyengar, C. Jorgensen, S. Rao, and C. Weisbin, "Learned navigation paths for a robot in unexplored terrain," in *Proc. 2nd Conf. Artificial Intell. Applicat.*, 1985, pp. 147-156.
[14] B. Kuipers and Y. Byun, "A qualitative approach to robot exploration and map-learning," in *Proc. Workshop Spatial Reasoning Multisensor Fusion*. Los Altos, CA: Morgan Kaufmann, 1987, pp. 390-404.
[15] B. Kuipers and T. Levitt, "Navigation and mapping in large-scale space," *AI Mag.*, Summer 1988, pp. 25-43.
[16] S. Lang and A. Wong, "Building geometric world models with graph synthesis for sensor fusion in mobile robots," *Computational Intel.*, vol. 6, pp. 91-107, 1990.
[17] L. Matthies and S. Shafer, "Error modeling in stereo navigation," *IEEE J. Robotics Automat.*, vol. 3, pp. 239-248, June 1987.
[18] H. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Mag.*, Summer 1988, pp. 61-74.
[19] N. Rao, S. Iyengar, and G. deSaussure, "The visit problem: Visibility graph-based solution," in *Proc. Inf. Conf. Robotics Automat.*, 1988, pp. 1650-1655.
[20] R. Smith, M. Self, and P. Cheeseman, "A stochastic map for uncertain spatial relationships," in *Proc. Workshop Spatial Reasoning Multisensor Fusion*, 1987.
[21] M. Turchan and A. Wong, "Low-level learning for a mobile robot: Environment model acquisition," in *Proc. 2nd Conf. Artificial Intell. Applicat.*, 1985, pp. 156-161.

Polynomial Time Collision Detection for Manipulator Paths Specified by Joint Motions

Achim Schweikard

Abstract—**An exact collision detection algorithm is described and analyzed. The time bound considers the complexity of the solids, the number of joints, and the number of distinct collision configurations. A bound for the number of collision configurations can be taken directly from the input data. The algorithm is based on an exact treatment of trigonometric expressions. The representation of trigonometric constants is discussed in an appendix.**

I. INTRODUCTION

The detection of intersections and collisions between objects is related to many geometric planning tasks. For example, collision detection methods are used as the basis of standard motion planning techniques. Clearly, in order to derive exact collision detection methods and asymptotic time bounds for collision detection, it is necessary to describe the complexity of given motions. The following problems are distinguished in this context.

1) Intersection detection: Given two objects in *fixed* configuration, test whether the objects have a point in common.
2) Intersection computation: Given two fixed objects, compute the intersection, i.e., the set of points common to the two objects.
3) Collision detection: Given objects *and* motions, decide whether an intersection will occur during the motions.

The problem considered here is an extension of the third problem: Given solids and motions, compute the time intervals in which an intersection will occur. Thus, common points are *not* computed. In what follows we will refer to a single motion of several objects, i.e., simultaneous motions of several objects are specified with respect to a single common time parameter t.

The problem of collision detection has been studied in many contributions [2], [4]-[6], [8], [9], [11]. Boyse [2] considers two polyhedra, one of which is fixed while the other polyhedron is either translated or rotated; this case is useful in motion planning applications [13]. In [5] and [9], arbitrary motions of polyhedra are considered and fast collision detection methods are derived using a discretization of the given motion; a discretization of a motion is a finite set of sample configurations that will occur during the motion. An intersection detection method is then applied to each sample configuration. Gilbert and Hong [8] describe an iterative collision detection method. With this method, it can be decided whether or not a collision will occur, but iterative root finding subalgorithms and a resolution parameter have to be used and time bounds considering the complexity of the motion cannot be derived. A fast method for computing the distance between polyhedra is given by Lin and Canny [11]. This method finds the closest pair of features (i.e., vertices, faces, edges) for fixed polyhedra and can be applied to collision detection using discrete sample configurations. In general, criteria for the choice of the sample configurations are not available. A method in [5], and [6] reduces collision detection for three-dimensional moving solids to intersection testing in higher dimensions. Here, extrusions of moving solids into an additional

Manuscript received September 19, 1990; revised July 17, 1991.
The author was with the Technische Universität Berlin, Berlin, Germany. He is now with the Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305.
IEEE Log Number 9103562.