An Efficient Algorithm

For

Identifying Objects Using Robot Probes

by

Kelly A. Lyons

and

David Rappaport

The Department of Computing and Information Science

Queen's University

Kingston, Ontario, Canada

January, 1989

Revised May, 1991

Introduction

A problem being studied in robotics is that of recognizing objects using robot finger probes. A finger probe is a directed line l aimed at an object ρ. The result is a contact point c which the probe makes with the object. Given a convex planar n-sided object, Cole and Yap show that $3n-1$ probes are necessary and $3n$ probes are sufficient to determine the shape of the n-gon[Cole87]. Bernstein has shown that $2n+2$ probes are sufficient to identify ρ given $\rho \in \Gamma$, where Γ is a set of m convex planar n-sided objects[Bern86]. Skiena improves on Bernstein's result and identifies a convex object from a finite set of such objects in $2n+1$ probes[Skie88]. Both results use $O(n)$ probes. In this paper, we show that if the objects have a fixed orientation and position, $m-1$ probes are necessary and sufficient to identify an object. Natarajan solves a slightly different problem[Nata87]. Given one n-sided object with n possible orientations, Natarajan shows that $n/2$ probes are necessary and n are sufficient to determine the orientation of the object. This result is similar to ours with $m=n$ and each object identical but oriented differently (we allow m different objects).

We present an algorithm that preprocesses a set of objects for efficient probing. We also present a probing scheme and algorithms for deleting objects from or inserting new objects into the set.

The number of probes used in the probing strategies of Bernstein and Skiena are independent of the cardinality of Γ. Their method identifies every edge of the object in sequence. On the other hand, the number of probes used in our method is independent of the number of sides of the objects. After no more than $m-1$ carefully chosen probes, the object is identified. Furthermore, the probes can be deployed in parallel and our strategy allows efficient insertion and deletion of objects.

The Results

We begin with some definitions and notation. Consider objects moving along a conveyor belt and resting against a vertical lip. Mathematically, we represent the conveyor belt as a plane which we call the *guide plane*, and the lip as a line L on the guide plane. Further, we distinguish a point p on L. The guide plane and lip are used by Natarajan to determine the orientation of a convex polygon[Nata87].

For each object $\rho \in \Gamma$, let $e[\rho]$ denote a specific edge so that $l[\rho]$ and $r[\rho]$ are its left and right endpoints, respectively. We assume that, before probing an object ρ, $e[\rho]$ is collinear with L and $l[\rho]$ is coincident with the point p.

The main result of this paper is the following theorem.

Theorem 1: Let Γ be a set of m n-sided convex polygonal objects and let L be a line on the guide plane. $m-1$ probes are both necessary sufficient to identify an object $\rho \in \Gamma$ positioned so that $l[\rho]$ is coincident at p, a point on L, and $e[\rho]$ is collinear with L.

We will prove necessity first. In the sequel, we will prove sufficiency.

Proof of Necessity:

We construct m convex polygons to show that any probing strategy must use at least $m-1$ probes to identify an object. We use a polar coordinate system (r, θ), where r is a distance and θ is a polar angle measured clockwise from 12 o'clock, to define points from which we construct the objects. Let $A = \{a_0, a_1, ..., a_m\}$, a set of $m+1$ points, where $a_i = (1, (2i+1)\pi/n)$ for $i = 0 .. m$, and let $B = \{b_1, b_2, ..., b_m\}$, a set of m points, where $b_i = (1/\cos(\pi/n), 2\pi i/n)$ for $i = 1 .. m$. Let the m objects be labeled ρ_i, for $i = 1 .. m$, and let ρ_i be constructed by taking the convex hull of $\{b_i\} \cup A$ which we will denote by $CH(\{b_i\} \cup A)$. Observe that the intersection of the objects is a regular n-gon $I = CH(A)$ and that the union of the objects is a convex polygon $U = CH(A \cup B)$. See figure 1.

The key to our proof relies on the fact that all object edges are either on the boundary of I or on the boundary of U and that both I and U are convex. To identify an object a probe must make contact with an object edge on U. Since U is convex, a single probe can identify at most one object. Furthermore, by the convexity of both I and U, a probe that makes contact with an edge on the boundary of I misses at most one object. So a single probe can distinguish at most one object from the rest.

Therefore, at least $m-1$ probes are necessary to identify an object.

End of Proof of Necessity

Sufficiency is shown by describing the preprocessing and identification algorithms.

The Preprocessing Algorithm

For preprocessing, we translate each of the m objects one at a time so that $l[\rho]$ is coincident at the point p and $e[\rho]$ is collinear with L, for all $\rho \in \Gamma$. Figure 2 displays $m=8$ objects as they are positioned after the completion of the preprocessing phase.

The preprocessing algorithm determines a set of probes that uniquely identifies each of the objects. Each probe begins outside the object and meets the object at a *contact point.* We represent each of these probes by a ray originating at the point p and extending through the contact point.

The preprocessing algorithm constructs a search tree that is used by the identification algorithm. We denote the tree constructed by the preprocessing algorithm as an *O-tree.* To identify an object, a path from the root to a leaf of the O-tree is traced. At each internal node of the O-tree, we store a ray and a contact point. All the rays originate at the point p and are represented by the

---

Figure 1:
A set of objects are shown to illustrate the proof of necessity. The white vertices represent the set A and the black vertices represent the set B. The intersection of the objects is a regular hexagon (shaded).

Figure 2:
$m=8$ objects in their specific position and orientation as they would be at the completion of the preprocessing step.

angle they make with the line L. The contact point is the point of intersection between a ray and a non-empty subset of Γ. Our strategy is to find a collection of rays so that probes along the rays aimed at p, uniquely identify each of the objects. To obtain our bound of $m-1$ probes we ensure that each ray makes a unique contact point with at least two distinct objects.

Our algorithm uses the following variables and data structures. The value $curr[\rho]$ denotes the current vertex of object ρ, and $prev[\rho]$ denotes the clockwise predecessor of $curr[\rho]$. Each node s in the O-tree contains fields labeled ray, $cont$, $object$, and $leaf$, as well as pointer fields to its children. The fields $s.ray$ and $s.cont$ store a ray direction and a contact point, respectively. $s.leaf$ is set to true if and only if s is a leaf node. If $s.leaf$ is true, then $s.object$ is an object label, otherwise, $s.object$ is undefined. We define a function $\theta(v)$ where v is a vertex of an object to denote the clockwise angle formed between the line pv and L. We will use the function *advance* to set $curr[\rho]$ to the clockwise successor of $curr[\rho]$ in ρ.

The preprocessing algorithm makes use of two procedures: AddObject which takes as parameters the root of the O-tree and an object ρ and adds ρ to the O-tree, computing a new ray direction if necessary; and GetNextRay which takes as parameters two objects ρ and ρ', and a node CN of the O-tree and finds a ray that distinguishes ρ from ρ', updating the O-tree at CN accordingly.

algorithm PREPROCESS
    {Initialize}
    root.leaf $\leftarrow$ false
    root.ray $\leftarrow$ the ray originating at p along L
    {Iteratively add object ρ_i to the O-tree.}
    for $i \leftarrow 1$ to m do
        call AddObject(root, ρ_i, $r[\rho_i]$)
    end for
end PREPROCESS

procedure AddObject(CN, ρ, I)
{CN is an internal node in the O-tree, ρ is the object being added, and I is the current vertex in ρ.}
Step 1:
if I ≠ s.cont for all children s of CN then
Create a child s′ of CN
s′.leaf ← true
s′.object ← ρ
s′.cont ← I
RETURN
Step 2:
{I = s.cont for some child s of CN.}
if s.leaf then
{We find a ray to distinguish s.object from ρ and make s the parent of the two leaf nodes.}
s.leaf ← false
{Find the two vertices of s.object in between which CN.ray intersects s.object.}
Find curr[s.object] such that CN.ray intersects s.object in the closed interval [prev[s.object], curr[s.object]].
call GetNextRay (ρ, s.object, s)
RETURN
Step 3:
else {node s is not a leaf}
{Find the two vertices of ρ in between which s.ray intersects ρ.}
Find curr[ρ] such that s.ray intersects ρ in the closed interval [prev[ρ], curr[ρ]]
i ← the intersection of s.ray and ρ
call AddObject(s, ρ, i)
end AddObject

procedure GetNextRay(ρ, ρ′, CN)
Step 1:
do while the curr[ρ] = curr[ρ′]
advance(curr[ρ])
advance(curr[ρ′])
$\bar{\theta}$ ← min{θ(curr[ρ]), θ(curr[ρ′])}
Step 2:
if curr[ρ], curr[ρ′] and prev[ρ](= prev[ρ′]) are all collinear and they are not all collinear with p then
{The ray originating at p of angle $\bar{\theta}$ contacts ρ and ρ′ at the same point. Recompute θ. (See figure 3)}
if θ(curr[ρ]) = $\bar{\theta}$ then
advance(curr[ρ])
else
advance(curr[ρ′])
$\bar{\theta}$ ← min{θ(curr[ρ]), θ(curr[ρ′])}
Step 3:
CN.ray ← the ray originating at p of angle $\bar{\theta}$
{Let I[ρ](I[ρ′]) be the intersection of CN.ray with ρ(ρ′).}
Step 4:
Create two children, c and c′ of CN

c.leaf ← true
c′.leaf ← true
c.cont ← I[ρ]
c′.cont ← I[ρ′]
c.object ← ρ
c′.object ← ρ′
end GetNextRay

Correctness of the Algorithm

Lemma 1:

GetNextRay finds a new ray that distinguishes ρ and ρ′; that is, $I[\rho] \neq I[\rho']$.

Proof of Lemma 1:

If the objects ρ and ρ′ are distinguishable then there is either a vertex $v \in \rho$ such that $v \notin \rho'$ or a vertex $v' \in \rho'$ such that $v' \notin \rho$. Step 1 of GetNextRay loops until a vertex of ρ differs from a vertex of ρ′ which will happen since ρ and ρ′ are distinguishable. Step 2 of GetNextRay check a condition and advances curr[ρ] and curr[ρ′], if necessary. Thus, the procedure reaches the beginning of step 3 where the following conditions hold: $curr[\rho] \neq curr[\rho']$ and curr[ρ], curr[ρ′] are not both on the same line with prev[ρ](= prev[ρ′]) (or curr[ρ] and curr[ρ′] are both on the same line with prev[ρ] and p). Consider triangles Δ(ρ) and Δ(ρ′) with vertices p, prev[ρ], and curr[ρ], and vertices p, prev[ρ′], and curr[ρ′], respectively. Any line passing through p and the interior or the boundary of Δ(ρ) and Δ(ρ′) intersects ρ in the closed interval [prev[ρ], curr[ρ]] and intersects ρ′ in the closed interval [prev[ρ′], curr[ρ′]]. The line l through p of angle θ is such a line, because $\theta = \min\{\theta(curr[\rho], \theta(curr[\rho'])\}$. Thus, we have $I[\rho] \neq I[\rho']$ because $curr[\rho] \neq curr[\rho']$. Figure 3 illustrates the situation at the beginning of step 3 of GetNextRay.

Figure 3:
At the beginning of step 3 of GetNextRay, we are in a state as illustrated by (a) or (b).

End of Proof of Lemma 1

Lemma 2:

The preprocessing algorithm calculates no more than $m-1$ ray directions.

Proof of Lemma 2:

Observe that one ray is computed for every internal node in the O-tree including the root. The number of internal nodes in the O-tree is bounded by the maximum possible number of internal nodes when the O-tree is a binary tree.

The number of internal nodes in a binary tree is one less than the number of leaf nodes. Since there are m objects in Γ, there are m leaf nodes in an O-tree constructed by the preprocessing algorithm; therefore, the maximum number of ray directions calculated by the preprocessing algorithm is $m-1$.

End of Proof of Lemma 2

Analysis of the Algorithm

Theorem 2:

Algorithm Preprocess has a worst case time complexity of $O(mn+m^2)$ and a space complexity of $O(mn)$.

Proof of Theorem 2:

Algorithm Preprocess performs some constant time initialization then calls procedure AddObject m times. Step 1 of AddObject takes $O(m)$ time to examine every child of the current node CN. Step 2 computes a new ray if ρ has the same intersection point as exactly one other object. Finding curr[s.object] and the call to GetNextRay uses no more than $O(n)$ operations, since both walk along the edges of an object without backtracking and each object has n edges.

If s is not a leaf node, step 3 of AddObject finds the intersection of s.ray with ρ and calls AddObject recursively. We find the intersection between s.ray and ρ by stepping around the vertices of ρ in a clockwise direction. At most every edge of ρ is tested once in this step over all calls to AddObject, since ρ is convex and all the rays are ordered around the objects in a clockwise direction. Therefore, step 3 takes $O(n)$ time over all calls to AddObject. The procedure AddObject has a worst case time complexity of $O(m+n)$, therefore, the algorithm Preprocess takes $O(mn+m^2)$ time. If $m<<n$ then the $O(mn)$ term dominates.

The space complexity of the preprocessing algorithm is $O(mn)$. Input to the algorithm is a set of m objects with n vertices each. The O-tree is stored by the preprocessing algorithm and there are $O(m)$ nodes in the O-tree. The items stored at each node take $O(1)$ space. Therefore, the space complexity is dominated by the size of the input, $O(mn)$.

End of Proof of Theorem 2

The Identification Algorithm

The preprocessing algorithm calculates $R \leq m-1$ ray directions. When an object, $\rho \in \Gamma$ travels along the conveyor belt and stops so that $l[\rho]$ is at the point p on L, R probes are sent along the ray directions. These probes are executed in any order or in parallel and the result is a set of R

corresponding contact points. The task of the identification algorithm is to take the set of contact points and probe directions and identify ρ.

The identification algorithm traverses the O-tree constructed by the preprocessing algorithm starting with the root. Let c be the contact point resulting from the probe aimed at p along root.ray. The identification algorithm traverses the child s of the root for which s.cont = c and so on down the tree until it reaches a leaf node. The object stored at that leaf node is identified as the object ρ.

The algorithm Identify is called with two arguments: CN, the current node of the O-tree and c, the contact point that a probe along CN.ray makes with ρ. Initially, CN $\leftarrow$ root.

algorithm IDENTIFY(CN, c)
{Examine each child CN.child of CN to find the child, s for which s.cont = c.}
if s.leaf then
RETURN s.object
else
call IDENTIFY(s, contact point corresponding to the probe along s.ray)

The complexity of algorithm Identify follows from the size of the O-tree. At each node CN, the identification algorithm examines each child of CN to find the intersection point that is equal to the given contact point. There are at most $2m-1$ or $O(m)$ nodes in the O-tree; therefore, the tree is traversed in $O(m)$ time.

Theorem 3 proves the correctness of the identification algorithm.

Theorem 3:

Algorithm Identify uniquely identifies an object from the set Γ.

Proof of Theorem 3:

Consider two distinct objects, ρ_i and ρ_j from Γ. Probes aimed at ρ_i and ρ_j along the ray directions calculated in the preprocessing algorithm result in two sets of contact points. Let C_i be the set of contact points resulting from the probes aimed at ρ_i and let C_j be the set of contact points resulting from the probes aimed at ρ_j. We show that $C_i \neq C_j$.

Let s_1 and s_2 be sibling nodes in the O-tree with the shared parent S. Let EC_1 be the equivalence class of objects in Γ intersected by S.ray at the point s_1.cont and let EC_2 be the equivalence class of objects in Γ intersected by S.ray at the point s_2.cont. s_1.cont is distinct from s_2.cont. Therefore, a probe along S.ray will contact those objects in EC_1 at at distinct point from those objects in EC_2.

We trace along the path from the leaf that holds ρ_i and along the path from the leaf that holds ρ_j to the node S, their closest common ancestor. ρ_i is in EC_i, the equivalence class of objects in Γ intersected by S.ray at s_i.cont where s_i is a child of S. ρ_j is in EC_j, the equivalence class of objects in Γ intersected by S.ray at s_j.cont where s_j is a child of S. $s_i \neq s_j$ since S is the closest common ancestor of ρ_i and ρ_j. Therefore, a probe along S.ray contacts ρ_i at a distinct point from ρ_j and $C_i \neq C_j$.

End of Proof of Theorem 3

Updating the Set of Objects

Suppose we have an application where the set of objects in Γ undergoes periodic changes. We can update the set of objects in $O(m+n)$ time. Adding a new object results in adding at most one new probe direction. Deleting an object results in deleting at most one probe direction.

In this section we show how an object can be inserted into the set and the O-tree updated in $O(m+n)$ time, and how an object can be deleted from the set and the O-tree updated in $O(m)$ time.

Adding an Object to the Set

If an object ρ_{m+1} is added to Γ, a new ray direction may be necessary to distinguish ρ_{m+1} from $\rho_1, \rho_2, \ldots, \rho_m$. We assume that $e[\rho_{m+1}]$ is known and collinear with L, and $l[\rho_{m+1}]$ is coincident at the point p on L before probing. The procedure AddObject used by the preprocessing algorithm can be used to add a new object to the set in $O(m+n)$ time.

Deleting an Object from the set

If an object ρ_i, $1 \le i \le m$ is removed from Γ, then one of the ray directions computed in the preprocessing algorithm may have to be removed in order to guarantee that no more than $m-1$ probes are used to uniquely identify each of the objects. The algorithm Delete Object runs in $O(m)$ time.

Consider additional links in the O-tree that connect the leaves of the tree to one another. Also consider back links from the leaf nodes to their parents and so on up the tree. These links can be established by traversing the O-tree once in $O(m)$ time.

We search the leaves of the O-tree for the leaf node s that holds ρ_i. The node s is removed from the O-tree. Let S be the parent of s. If s has exactly one sibling s′ and s′ is a leaf node, then s′ is removed from the O-tree, as well. S.ray is deleted from the node S since it no longer distinguishes any objects. The object that was stored at s′ is stored at S.

If s has exactly one sibling s′ and s′ is not a leaf node then S.ray is deleted. S is replaced by the node s′ and all of the descendents of s′ remain the same. The contact point stored at s′ is replaced by that stored at S.

If s has more than one sibling in the O-tree then s is removed from the O-tree and no ray is deleted.

Delete Object is called with the object ρ_i to be deleted.

algorithm DELETE OBJECT(ρ_i)
Step 1:
Search the leaf nodes of the O-tree until the node s that holds ρ_i is found.
Delete the leaf node s from the O-tree.
Step 2:
if s has more than one sibling then
RETURN
Step 3:
else {Let the one sibling of s be s′.}
{Let the parent of s be S.}
if s′ is a leaf node then
S.object ← s′.object
S.leaf ← true
Delete s′ from the O-tree.
Delete S.ray from S
RETURN
else {s′ is NOT a leaf node.}
s′.cont ← S.cont
Replace S with the node s′
{The descendents of s′ remain the same.}

RETURN
end else

The first step of the algorithm searches the leaf nodes of the O-tree in $O(m)$ time. Step 2 checks if s has more than one sibling in constant time since the number of siblings of s can be stored at s or at the parent of s. Deleting information stored in the O-tree and replacing one node with another in the O-tree takes constant time. Therefore, Delete Object is an $O(m)$ algorithm.

Conclusions

If fixing the position and orientation of the objects in the set is not feasible, then our strategy can still be used. However, more probes are needed and the processing is more costly. We allow each of the m objects ρ to be oriented so that any one of their n edges is $e[\rho]$. The left most vertex of $e[\rho]$ is at the point p. For each object, there are n such orientations. We treat each orientation as a different object; thus, there are mn objects in the set and a total of mn^2 vertices in the set. Our preprocessing algorithm is performed on the set of mn objects resulting in no more than $mn-1$ probe directions. The preprocessing algorithm takes $O(mn^2)$ time and uses $O(mn^2)$ space. The identification algorithm takes $O(mn)$ time. AddObject takes $O(mn)$ time and Delete Object takes $O(mn)$ time. We note that in applications where fixing the orientation of the objects is not feasible, the algorithms of Bernstein[Bern86] or Skiena[Skie88] are more appropriate for $m > 2$.

Our algorithm finds no more than $R \leq m-1$ probe directions; however, in some cases, $l < R$ probes are sufficient to identify each of the m objects. Recently, Belleville [Bell90] has shown that determining a minimum number of probes that identifies an object from a set of objects is NP-hard.

A problem for the future is to identify three dimensional objects in a similar manner. Consider a known face f lying on the conveyor belt or guide plane and an edge of f resting against the lip of the guide plane. Our strategy may be extendible to solve the problem in three dimensions.

Other future work involves developing a similar probing strategy for star-shaped polygonal objects.

Acknowledgements

The authors wish to thank an anonymous referee for pointing out a problem in our proof of Necessity and for helpful comments that allowed us to greatly simplify the proof of Theorem 2.

References

[Bell90]
Belleville, Patrice, *Probing polygons minimally is hard,* Manuscript, McGill University, 1990.

[Bern86]
Bernstein, Herbert J., *Determining the Shape of a Convex* $n-$*sided Polygon by Using* $2n+k$ *Tactile Probes,* Information Processing Letters, Vol. 22, 1986, 255-260.

[Cole87]
Cole, Richard, and Chee K. Yap, *Shape from Probing,* Journal of Algorithms, Vol. 8, 1987, 19-38.

[Lyon88]
Lyons, Kelly A., *An Efficient Algorithm for Identifying Objects Using Robot Probes,* MSc Thesis, Queen's University, December 1988.

[Nata87]
Natarajan, B.K., *On Detecting the Orientation of Polygons and Polyhedra,* Proceedings of the Third Annual Symposium on Computational Geometry, Waterloo, Ontario, June 1987.

[Skie88]
Skiena, Steven Sol, *Geometric Probing,* Technical Report UIUCDCS-R-88-1425, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1988.