from seven to three after the fifth trial. There was no evidence of learning in the voice-only mode, with the number of commands for a trial ranging between thirteen and nineteen across fourteen trials.

IV. Discussion

Several schemes have been suggested for automation of grasping using optoelectronic sensing. In general designs, several strategies including crossed focused IR beams have been used to obtain precise distance information [6]–[11]. In a space application, integrated sensors and signal processing electronics were placed in hollow gripper pads with signals apparently digitally multiplexed in the gripper pad [12]. In previous rehabilitation applications, twelve and sixteen optical sensors were used to facilitate gripper movement and grasping [4], [6]. A system with three gripper sensors has been proposed in another rehabilitation application [13].

The use of optoelectronic sensors to improve symmetric grasping is generally applicable. For example, sensor schemes such as the one implemented here should provide enhanced symmetric grasping capabilities for other robots without use of crossed focused beams. The key elements of the particular implementation were the use of standard components in simple configurations, the analog multiplexing of outputs from switched sensors at the gripper, and the use of a central computer with programs in a common language.

There were a number of advantages to using switched sensors and multiplexed sensor outputs. With only one LED on at a time, crosstalk between sensor pairs was eliminated. Having one signal line for sensor outputs provided the capability to add sensors with a minimum of additional components other than the sensor elements themselves. For example, since both the decoder and the analog multiplexer have a capacity of sixteen, expansion of the described system to have sixteen sensors instead of eight would involve using only one additional wire, that for control. Program modifications would, of course, also be needed to access and process the additional information.

V. Conclusions

Simple optoelectronic sensors on the gripper of a robot hand provided information which led to fewer voice commands required for symmetric grasping. Use of a central computer to process sensor information and to control the robot was made practical by time multiplexing the sensors. The symmetric-grasping capabilities and flexibility provided by the integration of voice control and optoelectronic sensors should prove to be useful in rehabilitation as well as other robot applications.

Acknowledgment

The authors wish to thank Dr. P. N. Hale, Jr., Center Director, for his support and encouragement during the project.

References

[1] M. LeBlanc and L. Leifer, "Environmental control and robotic manipulation aids," *IEEE Eng. Med. Biol. Mag.*, vol. 1, pp. 16–22, Dec. 1982.

[2] L. J. Leifer, "Robotic manipulation aids in rehabilitation," in *Technology for Independent Living II*, M. R. Redden and V. W. Stern, Eds. Washington, DC: Amer. Assoc. Advancement Sci., pp. 107–120, 1983.

[3] *Interactive Robotic Aids—One Option for Independent Living*, R. Foulds, Ed. New York, NY: World Rehabilitation Fund, 1986.

[4] H. S. Lee and L. J. Leifer, "Automated robotic grasping by use of optical proximity sensors," in *Proc. 10th Annu. Conf. on Rehabilitation Technology*, pp. 790–792, 1987.

[5] L. J. Leifer, R. Sun, and H. F. M. Van der Loos, "The need for a smart sensate hand in rehabilitative robotics," in *1980 Advances in Bioengineering*, V. C. Mow, Ed. New York, NY: ASME, 1980, pp. 285–288.

[6] ——, "A smart sensate hand for terminal device centered control of a rehabilitative manipulator," in *1980 Advances in Bioengineering*, V. C. Mow, Ed. New York, NY: ASME, 1980, pp. 69–72.

[7] T. Kanade and T. Sommer, "An optical proximity sensor for measuring surface position and orientation for robot manipulation," in *Robotics Research: The First International Symposium*, M. Brady and R. Paul, Eds. Cambridge, MA: MIT Press, 1984, pp. 547–563.

[8] C. A Rosen and D. Nitzan, "Use of sensors in programmable automation," *Computer*, vol. 10, pp. 12–23, Dec. 1977.

[9] A. K. Bejczy, "Effect of hand-based sensors on manipulator control performance," *J. Mechanism Machine Theory*, vol. 12, pp. 547–567, 1977.

[10] A. R. Johnston, "Proximity sensor technology for manipulator end effectors," *J. Mechanism Machine Theory*, vol. 12, pp. 95–109, 1977.

[11] B. Espiau, "Use of optical reflectance sensors," in *Recent Advances in Robotics*, G. Beni and S. Hackwood, Eds. New York, NY: Wiley, 1985, pp. 313–357.

[12] R. R. Killion, "Robot gripper with signal processing," NASA Tech. Support Package from JPL Invention Rep. NPO-16977/6491, Pasadena, CA, Jet Propulsion Lab., 1988.

[13] N. Valettas and A. A. Goldenberg, "The development of a robotic aid with automatic grasping capabilities," in *Proc. Int. Conf. Assoc. Advancement Rehabilitation Technol. (ICAART 88)*, pp. 464–465, June 1988.

On Some Link Distance Problems in a Simple Polygon

SUBHASH SURI

Abstract—**A technique is presented for preprocessing a simple polygon to answer link distance queries. The preprocessing requires linear time, plus the time to triangulate the polygon, and uses linear storage. As an application of the technique, optimal algorithms for several fundamental link distance problems are derived.**

I. Introduction

Consider the following motion planning problem: a point-size robot is to be navigated inside a simple polygon P such that the number of turns is minimized; robot moves along a polygonal path and the boundary of the polygon acts as an obstacle. This motivates the notion of link distance: the *link distance* between two points is the minimum number of edges in any polygonal path that connects the points without intersecting the boundary of P. In robot systems, the straight-line motion is frequently far cheaper compared to the rotation or turning. The link distance is an appropriate measure of distance for such systems. As a second application, consider a microwave communication network: the direct communication is possible only if the transmitter and the receiver are mutually "visible," otherwise special devices such as repeaters are needed to turn corners. Minimizing turns in such a situation has clear economic benefits.

In this communication, we present a general technique that can be used to solve a number of classical shortest path problems for the link measure of distance. The technique consists of a partitioning of the polygon into regions over which the link distance from a chosen source remains constant. The dual graph of this planar subdivision is a tree, which we call a *window tree*. The window tree is the central data structure of this communication, and we demonstrate its versatility by using it to solve numerous link distance problems.

Our first set of problems has the following flavor: given a fixed source x, preprocess P so that the link distance (or a minimum link

Manuscript received April 20, 1988; revised March 3, 1989. This research was supported by the National Science Foundation under Grant DCR-83-51468 and under grants from GM, IBM, and Martin-Marietta.

The author was with the Johns Hopkins University, Baltimore, MD. He is now with Bellcore, Morristown, NJ 07960.

IEEE Log Number 8929632.

path) to a query destination can be computed efficiently. We solve this problem in logarithmic time for the link distance query and in time proportional to the number of links in the path query. Using the window tree, we also derive a simpler linear-time algorithm for computing the link diameter of P within ± 2; the *link diameter* is the maximum link distance between any two points of P. We can also compute the link distance between two query points within ± 2 in logarithmic time. The last two problems are not of the "fixed source" type and thus are considerably harder to solve exactly. Our preprocessing technique, namely, the computation of the window tree, takes only linear time and space, provided that a triangulation of the polygon is given; currently, the best bound for triangulation is $O(n \log \log n)$. Our result improves upon several previous algorithms that required $O(n \log n)$ time, even if the triangulation were given [2], [9].

Section II introduces the basic definitions. The window trees are described in Section III. Relationship between the link distance and the distance in the window tree is explored in Section IV. Section V presents algorithms for fixed source problems. Sections VI and VII, respectively, discuss the approximation algorithms for the link diameter and the two-point queries. We conclude in Section VIII.

II. Preliminaries

The *link distance* between two points x and y, denoted $d_L(x, y)$, is the minimum number of edges in any polygonal path that connects x and y without intersecting the boundary of P. More generally, for two point sets X and Y, $d_L(X, Y)$ denotes the smallest link distance over all pairs $x \in X$ and $y \in Y$. A path $\pi_L(x, y)$ is called a *minimum link path* between x and y if $\pi_L(x, y)$ has exactly $d_L(x, y)$ edges. (Minimum link paths are generally not unique.)

Two points x and y are called *mutually visible* if the line segment xy lies completely within P. The visibility polygon of a point $x \in P$, denoted $V_P(x)$, is the set of all points that are visible to x. (We omit the subscript whenever the intended polygon is clear from the context.) In a slight generalization, a point y is *weakly visible* from a line segment s if y is visible from some point on s. The weak visibility polygon of a line segment s, denoted $V_P(s)$, is the set of all points weakly visible from s. The visibility polygon of a point can be computed in linear time (see [3], [5], [8], [14]), and the weak visibility polygon of a segment can be computed in linear time, provided that the input polygon is triangulated [5]; currently, the best bound for triangulation is $O(n \log \log n)$ [13].

We use $\pi_E(x, y)$ to denote the Euclidean shortest path between x and y in P; notice that, unlike a minimum link path, an Euclidean shortest path in a simple polygon is always unique. The union of the shortest paths from x to all the vertices of P is a planar tree (rooted at x), and it is called the *shortest path tree of P with respect to x.*

Fact 1 [5]: Let P be a simple triangulated polygon having n vertices. A shortest path tree of P with respect to a point $x \in P$ can be computed in $O(n)$ time.

Let $e = ab$ be an edge of P and let $e' = cd$ be a diagonal or an edge of P such that the (counterclockwise) ordering of the points is a, b, d, c. We say that the shortest path $\pi_E(a, c)$ is *outward-convex* if the convex angles formed by successive segments of this path with the directed line ab keep increasing. A symmetric definition holds for $\pi_E(b, d)$; see Fig. 1.

Fact 2 [5]: An interior point of e' is visible to e iff both $\pi_E(a, c)$ and $\pi_E(b, d)$ are outward-convex. Furthermore, if xy and wz are the two internal tangents of $\pi_E(a, c)$ and $\pi_E(b, d)$, then the visible portion of e' is delimited by the intersections of e' with the two lines xy and wz.

III. The Window Partition and Window Tree

Let x be a point or a line segment in P. Consider the visibility polygon of x, $V(x)$. The edges of $V(x)$ either are (part of) edges of P or they are chords of P. The edges of the latter variety are called *windows* of $V(x)$. The set of points at link distance one from x is precisely $V(x)$. The points of link distance two are the points in

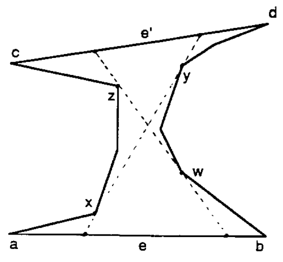

Fig. 1. Visibility polygon of an edge.

$P - V(x)$ that are visible from some window of $V(x)$. Repeatedly applying this procedure partitions P into regions of constant link distance from x.

Let c be a chord and let x be a line segment in P such that c and x do not intersect. The chord c divides P into two subpolygons, only one of which contains x. We use $P[c; x]$ to denote the subpolygon *not* containing x.

```
ALGORITHM Window-partition (x, P)
    Compute $V_P(x)$, the visibility polygon of $x$ in $P$;
    Let $w_1, w_2, \cdots, w_k$ be the windows of $V_P(x)$;
    for $i = 1$ to $k$ do Recursively call Window-partition
        ($w_i, P[w_i; x]$);
    Output $R(x) := V_P(x)$;
END
```

The algorithm *Window-partition (x, P)* returns a collection of regions that together partition P. Although each window of the partition is shared by two regions, it is natural to assign the points of a window to the region having the smaller link distance from x. The region associated with (or generated by) a window w is denoted $R(w)$. It is easy to see that the total size of the straight line planar graph induced by a window partition is $O(n)$.

Definition: The *window tree W(x)* denotes the planar dual of the window partition of P with respect to x: $W(x)$ has a node for each region of the partition and an arc between two nodes if their regions share an edge.

Each node of $W(x)$ is labeled with the window that generates the region corresponding to the node. Thus $W(x)$ is rooted at x. Fig. 2(a) shows a window partition, and Fig. 2(b) shows the corresponding window tree; in Fig. 2(a), bold numbers denote the link distance of x from any points in the region.

A naive implementation of the algorithm *Window-partition* will take $\Omega(n^2)$ time in the worst case; linear time per region in the partition. We show in the following that, by constructing the regions more carefully, we can compute the entire partition in optimal $O(n)$ time, provided that a triangulation of the polygon is given. The main idea behind the improved algorithm is this: given a triangulation of the polygon G, we will compute a visibility polygon V in time proportional to the number of triangles intersected by V (Section III-A). Further, we will show that a triangle of G intersects at most three regions of a window partition (Section III-B). These facts easily lead to the claimed bound.

A. A Fast Algorithm for Computing a Weak Visibility Polygon

The idea is to build shortest path trees from both end-points of the source line segment and construct visible portions of edges of P. To simplify the discussion, we describe how to compute the visibility polygon of an edge $e = ab$ of P; with minor modifications, the algorithm works for the chords as well. Let G denote a given triangulation of P. $V_P(e)$ is computed by calling a recursive procedure *VISIBILITY*. The procedure has two parameters: the first parameter is a diagonal of P, say, cd, and the second parameter is e. The algorithm outputs *visibility (cd; e)*, which roughly speaking is the part of $V(e)$ lying in $P[cd; e]$. To be more precise, *visibil-*

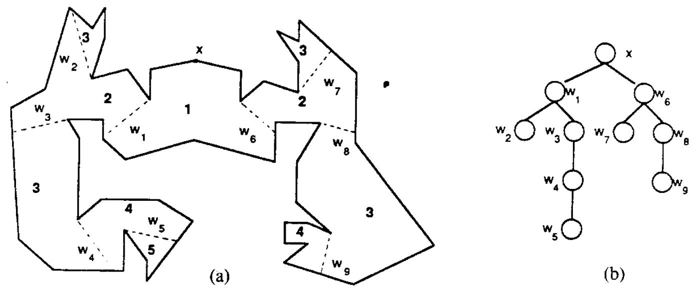

Fig. 2. A window partition and the corresponding window tree.

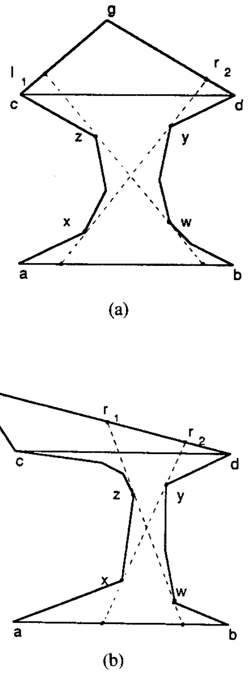

Fig. 3. Algorithm *VISIBILITY* $(cd;\ ab)$.

ity $(cd;\ e)$ is the *clockwise* sequence of edges of $V(e)$ contained in $P[cd;\ e]$; if cd is not completely visible to e, then (at most two) chords that "stitch" together parts of $V(e)$ lying on opposite sides of cd are also output. See Fig. 3 for the following procedure.

```
ALGORITHM VISIBILITY (cd; e)
1) Let $xy$, $wz$ be the internal tangents of the geodesic pair $\pi_E(a, c)$ and $\pi_E(b, d)$, where $x$, $z$ lie on $\pi_E(a, c)$ and $w$, $y$ lie on $\pi_E(b, d)$.
2) Let $\Delta cdg$ be the unique triangle in $G$ that has $cd$ as an edge and that has not been processed yet.
3) if $\pi_E(a, c)$ and $\pi_E(b, g)$ are both outward-convex then
     if $cg$ is an edge of $P$ then
       Let $l_1$, $l_2$ be the points where $cg$ intersects the lines $wz$ and $xy$, respectively;
       if $l_2$ is undefined[1] then Set visibility $(cg; e) := \{zl_1, l_1g\}$;
       else Set visibility $(cg; e) := \{zl_1, l_1l_2, l_2y\}$;
     else (*$cg$ is a diagonal of $P$*)
       Recursively call VISIBILITY $(cg; e)$ to compute visibility $(cg; e)$;
   else visibility $(cg; e) := \emptyset$;
4) if $\pi_E(a, g)$ and $\pi_E(b, d)$ are both outward-convex then
     if $gd$ is an edge of $P$ then
       Let $r_1$, $r_2$ be the points where $gd$ intersects the lines $wz$ and $xy$, respectively,
       if $r_1$ is undefined then Set visibility $(gd; e) := \{gr_2, r_2y\}$;
       else Set visibility $(gd; e) := \{zr_1, r_1r_2, r_2y\}$;
     else (*$gd$ is a diagonal of $P$*)
       Recursively call VISIBILITY $(gd; e)$ to compute visibility $(gd; e)$;
   else visibility $(gd; e) := \emptyset$;
5) Set visibility $(cd; e) :=$ visibility $(cg; e) \cup$ visibility $(gd; e)$.
END
```

In Fig. 3(a), both Steps 3 and 4 are performed. In Fig. 3(b), only Step 4 is performed because $\pi_E(b, g)$ is not outward-convex. If Δabu is the triangle in G with $e = ab$ as an edge, then we recursively call *VISIBILITY* $(au;\ e)$ and *VISIBILITY* $(ub;\ e)$. The final answer is computed as

$$V(e) = \textit{visibility}\,(au;\ e) \cup \textit{visibility}\,(ub;\ e).$$

To prove the correctness of the algorithm, consider an edge or a diagonal cg. If either $\pi_E(a, c)$ or $\pi_E(b, g)$ is not outward-convex, then neither cg nor $P[cg;\ e]$ is visible from e, and the algorithm correctly reports *visibility* $(cg;\ e) = \emptyset$. Otherwise, if cg is an edge, the algorithm returns the visible portion of cg and, if cg is a diagonal, it calls *VISIBILITY* $(cg;\ e)$; the correctness follows from Fact 2 in the former case and from induction in the latter.

To analyze the time complexity, consider the procedure call *VISIBILITY* $(cd;\ e)$. If g is the third vertex of the (yet unprocessed) triangle in G having cd as a diagonal, then we extend the shortest path trees from a and b to include g.[2] The cost of processing the triangle Δcdg equals the work involved in extending the two shortest path trees to g. Since we only extend the trees to the vertices of the triangles that are at least partially visible from e, the total time for computing $V(e)$ is $O(k_e)$, where k_e is the number of triangles intersected by $V(e)$ (see [5] for details of the shortest path algorithm). The auxiliary information such as whether the paths $\pi_E(a, c)$ and $\pi_E(b, g)$ are outward-convex and the internal tangents needed by the algorithm can also be maintained easily within this time. This completes the analysis of the algorithm *VISIBILITY*.

Remark: The algorithm is easily modified for computing visibility polygon of a chord; we only need to ensure that the subpolygon cut off by the chord is properly triangulated. One way to achieve this is to retriangulate the region consisting of the triangles intersected by the chord. Since this region is visible weakly, it can be triangulated in linear time (see Toussaint and Avis [1]).

The following lemma has thus been established:

[1] We say l_2 is undefined if the line determined by xy does not intersect the segment cg.

[2] We remark that we are simply executing the shortest path algorithm of [5] in a "controlled" fashion. The algorithm of [5] builds the tree by following the adjacent triangles in the triangulation. The algorithm *VISIBILITY* exercises its control by deciding when to extend the shortest path tree, when to stop, and which vertex to go to next. Specifically, we only want to visit triangles that are at least partially from e.

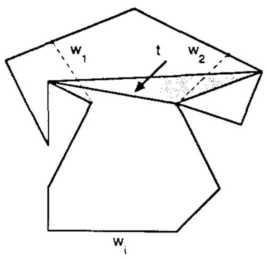

Fig. 4. Intersection of a triangle with three regions.

Lemma 1: Let P be a simple polygon of n vertices and let G be a given triangulation of P. Let s be a chord of P and let $P_1 \subseteq P$ be one of the two subpolygons that results from cutting P along s. Then, the visibility polygon of s in P_1 can be computed in time proportional to the number of triangles in G that are intersected by the visibility polygon.

B. Computing the Window Partition in Linear Time

Assume that P is triangulated; the triangulation cost is incurred once. We start by computing $V(x)$, the visibility polygon of x. This takes time $O(n)$ time [5]. Subsequently, each visibility polygon V is computed for a chord of P, which can be done in time proportional to the number of triangles of G intersected by V (Lemma 1). To prove the linear upper bound on the time complexity, we show in the following that a triangle of G intersects only a constant number of regions in the partition.

Let *Window-partition* $(w_i, P[w_i;\, w_j])$ be the first instance when a triangle $t \in G$ is processed; see Fig. 4. If t is entirely contained in $V_{P[w_i;\, w_j]}(w_i)$, then t cannot belong to any other region of the partition, and the claim obviously holds. Otherwise, the region $t \cap V_{P[w_i;\, w_j]}(w_i)$ is connected and is bounded by at most two windows of $V_{P[w_i;\, w_j]}(w_i)$, say, w_1 and w_2. Therefore, t is entirely contained in the union of the following three polygons: 1) the visibility polygon of w_i in $P[w_i;\, w_j]$, 2) the visibility polygon of w_1 in $P[w_1;\, w_i]$, and 3) the visibility polygon of w_2 in $P[w_2;\, w_i]$.

This proves the following theorem:

Theorem 1: Let P be a simple polygon and let x be an arbitrary point or a line segment in P. The window partition of P with respect to x can be constructed in triangulation plus a linear amount of time and linear space.

In the next section we establish a fundamental connection between the window trees and the link distance.

IV. Window Tree and Link Distance

Let T be a finite tree, rooted at r. The *tree distance* between two nodes $x, y \in T$ is the number of tree edges in the shortest path between x and y in T. The *height* of T is the maximum tree distance from the root r to any leaf of T. The *depth* of a node $x \in T$ is the tree distance of x from the root r.

Lemma 2: Let w be a node at depth k in the window tree $W(x)$, where x is a point or a line segment in P. Then $d_L(x, w) = k$, and a minimum link path $\pi_L(x, w)$ can be found in time $O(k)$.

Proof: Let $(x = w_0, w_1, \cdots, w_k = w)$ be the path from x to w in $W(x)$. Consider a minimum link path $\pi_L(x, w)$ between x and w. This path crosses each of the windows $w_1, w_2, \cdots, w_{k-1}$. Since w_i is invisible from any point in $R(w_j)$, for $j > i$, $\pi_L(x, w)$ has at least one vertex in each $R(w_i)$, for $i = 0, 1, \cdots, k-1$. This proves $d_L(x, w) \geq k$. To prove the converse, let $x_i \in w_i$ be the point of intersection between the lines determined by w_{i+1} and w_i, for $0 \leq i \leq k-1$, and let x_k be any point on $w = w_k$. Then, $(x_0, x_1, \cdots, x_k)$ is a k-link path connecting x and w. Therefore, $d_L(x, w) \leq k$ and the proof is finished. □

Given a point $p \in P$, let *wmate* (p) denote the (unique) node in $W(x)$ such that p lies in $R(wmate(p))$; we call $wmate(p)$ the *window mate* of p. Since p is weakly visible from the chord $wmate(p)$, the following is an immediate corollary of Lemma 2.

Lemma 3: Let p be an arbitrary point of P and let $wmate(p)$ be the window mate of p in $W(x)$. Then, $d_L(x, p) = k_p + 1$, where k_p is the depth of $wmate(p)$ in $W(x)$.

In the next section, we solve minimum-link path problems for a fixed source.

V. Solving Minimum-Link Path Problems

The main result of this section is stated in the following theorem:

Theorem 2: Given a triangulated simple polygon P of n vertices and a point $x \in P$, we can preprocess P in linear time, following which
1) we can compute the link distance between x and a query point y in time $O(\log n)$,
2) we can compute a minimum link path between x and a query point y in time $O(k + \log n)$, where $k = d_L(x, y)$,
3) we can construct a minimum link path from x to any vertex of P in time proportional to the number of edges in the path.[3]

Proof: We compute the window tree $W(x)$; this takes linear time. We also store with each node its depth in $W(x)$. To solve problem 1), we only need to compute the window mate of y: $d_L(x, y) = k_y + 1$, where k_y is the depth of $wmate(y)$. This can be accomplished in time $O(\log n)$ by a point-location technique for determining which region of the window partition contains y (see [4], [6]).

Once we have determined $wmate(y)$, the problem 2) can be solved easily by finding a point z on the chord $wmate(y)$ that is visible from y. This can be done in additional $O(\log n)$ time by using a "grounded visibility" result from [5]. The path $\pi_L(x, y)$ is obtained by appending the segment zy to the path $\pi_L(x, wmate(y))$. Since the latter can be computed in time $O(k_y)$ (Lemma 2), problem 2) can be solved in time $O(k_y + \log n)$.

To solve problem 3), we compute for each node $w \in W(x)$ the point where the line determined by w intersects $parent(w)$. This gives, in linear time, an implicit representation of minimum link paths from x to all the windows of the window partition of P. Next, we need, for each vertex $v \in P$, a point p_v on $wmate(v)$ that is visible from v. We can compute such points p_v during the construction of visibility polygons: while computing the visibility polygon of the window w, for each vertex v that is visible to w, we store a point $p_v \in w$ seen by v. (Specifically, let a and b be the endpoints of w and let $v \in R(w)$ be a vertex. If uv is the last edge of $\pi_E(a, v)$, then we can choose p_v to be the point where the line determined by uv intersects w.) To retrieve a minimum link path between x and a vertex $v \in P$, let $(x = w_0, w_1, \cdots, w_k = wmate(v))$ be the path from x to $wmate(v)$. If $x_i \in w_i$ is the point of intersection of the lines determined by w_{i+1} and w_i, for $0 \leq i \leq k-1$, then $\pi_L(x, v) = (x_0, x_1, \cdots, x_{k-1}, p_v, v)$. This path can be computed in linear time, using the stored pointers. This completes the proof. □

In the next section we consider the application of window trees to approximating link diameter and link-furthest neighbors.

VI. Approximating Link Diameter and Link-Furthest Neighbors

The link diameter of P, $D(P)$, is the maximum link distance between any two points of P. A *link-furthest* neighbor of a point x is a point $y \in P$ whose link distance from x is maximum. We let $\phi_L(x)$ denote the set of all link-furthest neighbors of a point x; in general, $\phi_L(x)$ is an infinite set. However, it is easily shown that $\phi_L(x)$ always contains a vertex of P. Therefore, the link diameter of P can be always realized between two vertices of P.

A link-furthest neighbor of a point v can be found in $O(n)$ (plus triangulation) time by constructing the window tree with respect to v. This leads to an $O(n^2)$ time algorithm for computing $D(P)$; the

[3] That is, if the queries are vertices of the polygon, then we need not pay logarithmic part of the cost in (2).

triangulation cost is incurred only once. An $O(n \log n)$ time algorithm for computing the link diameter of P has been obtained by Suri [11]. The algorithm in [11] however is fairly complicated and requires many different steps. In this section, we present a simple algorithm for approximating the link diameter within ± 2 in linear (plus triangulation) time. More precisely, for each vertex $u \in P$, we compute a number $\alpha(u)$ such that

$$d_L(u, y) - 2 \le \alpha(u) \le d_L(u, y) + 2$$

where $y \in \phi_L(u)$ is a true link-furthest neighbor of u. Thus $\alpha(u)$ is an approximation of the link distance between u and its link-furthest neighbors. We first reduce our problem to a furthest neighbors problem in a tree, and then show how to solve the latter.

A. Reduction to the Furthest Neighbors Problem on a Tree

Let $W(x)$ be the window tree of P with respect to an arbitrary vertex x. Let $g(y, z)$ denote the tree distance between the nodes y and z in $W(x)$.

Lemma 4: Let w_1, w_2 be two nodes of $W(x)$, and let $u_1 \in R(w_1)$, $u_2 \in R(w_2)$ be two points of P. Then $g(w_1, w_2) - 1 \le d_L(u_1, u_2) \le g(w_1, w_2) + 3$.

Proof: Let δ_1 and δ_2, respectively, be the tree distances of w_1 and w_2 from their lowest common ancestor w_0. Then $g(w_1, w_2) = \delta_1 + \delta_2$. By Lemma 3, $d_L(u_1, w_0) = \delta_1 + 1$ and $d_L(u_2, w_0) = \delta_1 + 1$, which readily implies $d_L(u_1, u_2) \le (\delta_1 + \delta_2 + 3)$.

To prove the lower bound, assume that $\delta_1, \delta_2 \ge 1$; otherwise, the proof is easy. Let w_1', w_2' be the children of w_0 whose respective subtrees contain w_1 and w_2. Then $g(w_1, w_1') = \delta_1 - 1$ and $g(w_2, w_2') = \delta_2 - 1$. Therefore, a path from u_1 (resp., u_2) with at most $\delta_1 - 1$ (resp., $\delta_2 - 1$) links lies entirely in $P[w_1'; x]$ (resp., $P[w_2'; x]$). The disjointness of $P[w_1'; x]$ and $P[w_2'; x]$ implies that $\pi_L(u_1, u_2)$ has at least $(\delta_1 - 1) + (\delta_2 - 1) + 1 = (\delta_1 + \delta_2 - 1)$ edges. This completes the proof.

□

A *furthest neighbor* of a node $y \in W(x)$ is another node z whose tree distance from y is maximum.

Lemma 5: Let $w_1 \in W(x)$ be a node and let w_2 be a furthest neighbor of w_1. Then, for all vertices $u_1 \in R(w_1)$, $\alpha(u_1) = g(w_1, w_2) + 1$ is a valid approximation.

Proof: Let u_0 be a link-furthest neighbor of u_1 and let w_0 be the window mate of u_0. By the previous lemma, $g(w_1, w_0) - 1 \le d_L(u_1, u_0) \le g(w_1, w_0) + 3$. Consider a vertex $u_2 \in R(w_2)$. Then $g(w_1, w_2) - 1 \le d_L(u_1, u_2)$. Since $d_L(u_1, u_2) \le d_L(u_1, u_0)$, we obtain

$$g(w_1, w_2) - 1 \le d_L(u_1, u_0).$$

Finally, $d_L(u_1, u_0) \le g(w_1, w_2) + 3$ holds because $g(w_1, w_0) \le g(w_1, w_2)$ by the hypothesis of the lemma. These bounds on $d_L(u_1, u_2)$ imply that the choice of $\alpha(u_1) = g(w_1, w_2) + 1$ guarantees

$$d_L(u_1, u_0) - 2 \le \alpha(u_1) \le d_L(u_1, u_0) + 2.$$

The lemma follows.

□

Next, we show that the furthest neighbors in a tree can be computed in linear time.

B. An Algorithm for Solving the Furthest Neighbors Problem on Trees

Let T be a (finite) rooted tree of n nodes. For each node $w \in T$, we want to compute $g_{\max}(w)$, which is the maximum tree distance between $w \in T$ and any other node of T. Let $g_1(w)$ (resp., $g_2(w)$) denote the maximum tree distance between w and a *descendant* (resp., *nondescendant*) node of w. Notice that $g_{\max}(w) = \max\{g_1(w), g_2(w)\}$. Once we observe that $g_1(w) = height(w)$, computing $g_1(w)$'s is easy: if $w_1, w_2, \cdots, w_k$ are children of w, then

$$g_1(w) = 1 + \max\{g_1(w_1), g_1(w_2), \cdots, g_1(w_k)\}.$$

Thus $g_1(w)$'s can be computed for all nodes of T by traversing the tree in postorder. This takes linear time [12]. $g_2(w)$'s also are computed recursively as follows:

1) $g_2(w) = 0$, if w is the root of T,
2) $g_2(w) = 1 + \max\{g_2(parent(w)), 1 + \max\{g_1(w_1), \cdots, g_1(w_{i-1}), g_1(w_{i+1}), \cdots, g_1(w_k)\}\}$ otherwise, where $w_1, w_2, \cdots, w_{i-1}, w_{i+1}, \cdots, w_k$ are siblings of w.

We can compute all $g_2(w)$'s by a second postorder traversal of T. In summary, we can compute approximate link-furthest neighbors for all vertices of P by solving the furthest neighbors problem in a window tree $W(x)$. This leads to the following theorem:

Theorem 3: The link diameter of a simple polygon can be computed within ± 2 in triangulation plus linear time.

Finally, we consider the problem of approximately determining the link distance between two query points. We show that, having preprocessed P with respect to some fixed point, the link distance between two query points can be determined within ± 2 in $O(\log n)$ time.

VII. Approximating Link Distance Between Two Query Points

We compute the window tree of P with respect to an arbitrary vertex x. Given two query points y and z, we determine in $O(\log n)$ time their window mates w_1 and w_2 [4], [6]. Next, we compute the lowest common ancestor of w_1 and w_2; this takes $O(1)$ time by a result of Harel and Tarjan [15]. By Lemma 4, $d_L(y, z)$ is within ± 2 of $g(w_1, w_2) + 1$.

Theorem 4: Given a simple polygon P of n vertices, we can preprocess P in triangulation plus linear time, following which the link distance between two query points can be determined within ± 2 in $O(\log n)$ time.

VIII. Conclusions

We presented an optimal algorithm for partitioning a triangulated polygon into regions of constant link distance from a fixed source (point or line segment). The technique led to optimal algorithms for several basic link distance problems. Indeed, several other applications of the window tree are possible, and can be found in [7], [16], and [17]. In [7], window trees are used to obtain an approximate link center of a polygon in $O(n \log n)$ times; the *link center* of P is the locus of points from which the maximum link distance to any point of P is minimized. Later, Ke [17] used window trees to obtain an $O(n \log n)$ time algorithm for computing the link center exactly.

In an altogether different development, Aggarwal, Moran, Shor, and Suri [16] use the window partitioning to find a closest visible vertex pair between two simple polygons; here the distance measure is Euclidean. The algorithm in [16] uses the window partitioning to decompose the region between the two input polygons into subregions such that a subregion only sees its neighboring regions. We believe that window trees will find further applications in solving computational geometry problems.

Of course, a natural (and perhaps more realistic) extension of this work will be to explore link distance problems in multiply connected regions. The problem seems hard, and some direct approaches may fail due to inherent complexity of the visibility polygons in a multiconnected region. It has been shown by Suri and O'Rourke [18] that the region visible from a line segment inside a multi-connected polygon of n vertices may have $\Omega(n^4)$ disconnected pieces. Another difficulty arises due to overlaps that will be caused by visibility polygons being computed from different directions. Nevertheless, we hope that the results of this communication will provide motivation and some insight into the general problem. The area of three dimensions is also hitherto unexplored as far as link distance is concerned.

Acknowledgment

The author wishes to thank J. O'Rourke for many pleasant discussions.

References

[1] G. T. Toussaint and D. Avis, "On a convex hull algorithm for polygons and its applications to triangulation problems," *Pattern Recogn.*, vol. 15, no. 1, pp. 23-29, 1982.

[2] H. ElGindy, "Hierarchical decomposition of polygons with applications," Ph.D. dissertation, School of Computer Science, McGill University, Montreal, P.Q., Canada, 1985.

[3] H. ElGindy and D. Avis, "A linear algorithm for computing the visibility polygon from a point," *J. Algorithms*, vol. 2, no. 2, pp. 186-197, 1981.

[4] H. Edelsbrunner, L. Guibas, and J. Stolfi, "Optimal point location in monotone subdivisions," *SIAM J. Comput.*, vol. 15, no. 2, pp. 317-340, 1986.

[5] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, "Linear-time algorithms for visibility & shortest path problems inside a triangulated simple polygon," *Algorithmica*, vol. 2, pp. 209-233, 1987.

[6] D. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM J. Comput.*, vol. 12, no. 1, pp. 28-35, 1983.

[7] W. Lenhart, R. Pollack, J. Sack, R. Seidel, M. Sharir, S. Suri, G. T. Toussaint, S. Whitesides, and C. Yap, "Computing the link center of a simple polygon," *Discrete Computat. Geom.*, vol. 3, no. 3, pp. 281-293, 1988.

[8] A. A. Melkman, "On computing the visibility polygon from a point," Tech. Rep. CS-85-260, Math. Comput. Sci. Dep., Ben Gurion Univ., Ber-Sheva, Israel, 1985.

[9] J. Reif and J. A. Storer, "Minimizing turns for discrete movement in the interior of a polygon," Tech. Rep., Harvard Univ., Cambridge, MA, Dec. 1985.

[10] S. Suri, "A linear time algorithm for minimum link paths inside a simple polygon," *Comput. Vision, Graph. Image Process.*, vol. 35, pp. 99-110, 1986.

[11] ——, "Minimum link paths in polygons and related problems," Ph.D. dissertation, Dep. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, Aug. 1987.

[12] R. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146-160, 1972.

[13] R. Tarjan and C. Van Wyk, "An $O(n \log \log n)$ time algorithm for triangulating simple polygons," *SIAM J. Comput.*, vol. 17, pp. 143-178, 1988.

[14] D. T. Lee, "Visibility of a simple polygon," *Comput. Vision, Graph. Image Process.*, vol. 22, pp. 207-221, 1983.

[15] D. Harel and R. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM J. Comput.*, vol. 13, no. 2, pp. 338-355, 1984.

[16] A. Aggarwal, S. Moran, P. Shor, and S. Suri, "Finding minimum visible vertex distance between two simple polygons in linear plus triangulation time," submitted for publication, 1988.

[17] Y. Ke, "An efficient algorithm for link distance problems inside a simple polygon," Tech. Rep., Johns Hopkins Univ., Baltimore, MD, June 1988.

[18] S. Suri and J. O'Rourke, "Worst-case optimal algorithms for constructing visibility polygons with holes,"in *2nd Annu. ACM Symposium on Computational Geometry*, pp. 14-23, 1986.

An Efficient Motion Control Algorithm for Robots with Wrist Singularities

JIM Z. C. LAI AND DANIEL YANG

Abstract—**An efficient algorithm is developed for the motion control of manipulators in the neighborhood of wrist singularity. The method, based on a scheduling scheme, guarantees a robot's location accuracy, maintains a desired Cartesian motion speed, and does not involve the complex computations of a robot's Jacobian and its inverse. In case studies, three algorithms, namely the maximum joint rates [1], the iterative method [1], and the proposed scheduling method are compared. We feel that the combination of simple computation, acceptable orientation error, and the ability of maintaining desired speed should make this method very attractive in practical application.**

Manuscript received May 24, 1988; revised March 6, 1989.

The authors are with the Department of Mechanical, Aerospace, and Nuclear Engineering, University of California, Los Angeles, Los Angeles, CA 90024.

IEEE Log Number 8931957.

I. Introduction

The inverse kinematics and singularity problem are two major concerns in robot's trajectory planning and control. For this reason, most industrial robots are so designed that their closed-form inverse kinematics exist. However, even for a well-defined robot, in certain configurations, the robot will be degenerated (or singular) and lose one or more degree of freedom. Paul [4] classified degeneracy of robots with spherical wrists into two parts: regional structure degeneracy and wrist degeneracy. Regional structure degeneracy can be avoided by restricting the work area. However, the wrist degeneracy is a problem which occurs virtually any place inside the workplace.

Mathematically, when a robot is singular, the determinant of the robot's Jacobian equals zero, and the inverse Jacobian does not exist. Let $J(\bar{\theta})$ be the Jacobian of a robot, then in singularity we have

$$\delta\bar{x} = J(\bar{\theta})\,\delta\bar{\theta} \qquad \text{and} \qquad \det(J(\theta)) = 0 \tag{1}$$

where $\delta\bar{x}$ and $\delta\bar{\theta}$ are the respective differential changes of Cartesian and joint coordinates. In this case, motions in certain directions will require some joint velocities to exceed the hardware's limits. To overcome this difficulty, most current techniques in practice command the robot to slow down so that all joint velocities remain within their velocity limits. This approach introduces severe vibrations for great decelerations are required. Nakamura and Hanafusa [2] used the SR-inverse (singularity robust inverse) of a Jacobian matrix to determine the approximate motion. Later Paul and Aboaf [4] proposed a modified Jacobian method to solve the problem of singularity and it is useful for the robot with a spherical wrist. Another method, referred here as the maximum joint rate method, is to set the velocities of those joints which are required to exceed their hardware limits to their maximum values. However, in this case, both the location and orientation errors will be large.

Due to the fact that for certain applications, like welding and spray painting, location accuracy is more important than that of the orientation, Lai and Menq [1] proposed an iterative method to achieve a required location with an acceptable level of orientation error. In this communication, a more efficient method is developed for the same purpose. This algorithm uses a scheme which chooses an adequate location ahead on the predefined path so that the originally desired velocity will be maintained, joint velocities will not exceed their hardware limits, and the position accuracy will be guaranteed. The only derivation from the predefined path will be the hand orientation. Yet this derivation will be small. In addition, the algorithm does not include the complex computation of Jacobian and its inverse.

II. Geometric Description

A manipulator with n joints in series can be schematically represented by Fig. 1. The Denavit-Hartenberg convention for link geometry representation is adopted. The geometric relationship between the coordinate frames at joints k and $k+1$ is described by the following 4×4 matrix:

$$A_k = \begin{bmatrix} \cos\theta_k & -\sin\theta_k \cos\alpha_k & \sin\theta_k \sin\alpha_k & a_k \cos\theta_k \\ \sin\theta_k & \cos\theta_k \cos\alpha_k & -\cos\theta_k \cos\alpha_k & a_k \sin\theta_k \\ 0 & \sin\alpha_k & \cos\alpha_k & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where θ_k, d_k, a_k, and α_k are the rotational angle, axial offset, common normal, and the twist angle, respectively. The location and orientation of the end-effector of a robot with six joints can be defined