where $\boldsymbol{a}_i$ are n-dimensional column vectors. The Frobenius norm of $\boldsymbol{A}$ is defined by

$$||\boldsymbol{A}||_F^2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2} = \sqrt{\sum_{i=1}^{m} ||\boldsymbol{a}_i||_2^2}.$$

Premultiplying with $\boldsymbol{Q}$ gives

$$\boldsymbol{QA} = [\boldsymbol{Qa}_1 \quad \boldsymbol{Qa}_2 \quad \cdots \quad \boldsymbol{Qa}_m]$$

where $\boldsymbol{Qa}_i$ are n-dimensional column vectors. This gives

$$\begin{aligned}||\boldsymbol{QA}||_F^2 &= ||\boldsymbol{Qa}_1||_2^2 + ||\boldsymbol{Qa}_2||_2^2 + \cdots + ||\boldsymbol{Qa}_m||_2^2 \\ &= ||\boldsymbol{a}_1||_2^2 + ||\boldsymbol{a}_2||_2^2 + \cdots + ||\boldsymbol{a}_m||_2^2 = ||\boldsymbol{A}||_F^2\end{aligned}$$

which means that the Frobenius norm is invariant under orthogonal transformations.

A similar argument gives

$$||\boldsymbol{QAZ}||_F = ||\boldsymbol{A}||_F$$

where $\boldsymbol{Z}$ is an orthogonal m-dimensional matrix.

This gives together with the the singular value decomposition (see [5]) for the matrix $\boldsymbol{A}$

$$||\boldsymbol{A}||_F = ||\boldsymbol{U\Sigma V}^{\mathrm{T}}||_F = ||\boldsymbol{\Sigma}||_F = \sqrt{\sum_{i=1}^{p} \sigma_i^2(\boldsymbol{A})}$$

where $p = \min\{m, n\}$ and $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices of appropriate dimensions. $\boldsymbol{\Sigma}$ is an $n \times m$ matrix where the upper left $p \times p$ submatrix is diag$[\sigma_1 \quad \sigma_2 \quad \cdots \quad \sigma_p]$ and the rest of the matrix is zero.

From the above it follows that, for the 3×6 matrix $\boldsymbol{R}$, the Frobenius norm is

$$||\boldsymbol{R}||_F = \sqrt{\sum_{i=1}^{3} \sigma_i^2(\boldsymbol{R})}.$$

References

[1] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *ASME J. Dynam. Syst., Meas., Contr.*, vol. 108, pp. 163–171, Sept. 1986.

[2] C. W. Wampler II, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares method," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-16, no. 1, pp. 93–101, Jan./Feb. 1986.

[3] A. A. Maciejewski and C. A. Klein, "Numerical filtering for the operation of robotic manipulators through kinematically singular configurations," *J. Robotic Syst.*, vol. 5, no. 6, pp. 527–552, 1988.

[4] G. Strang, *Linear Algebra and its Applications*, 2nd ed. New York: Academic, 1980.

[5] G. H. Golub, C. F. Van Loan, *Matrix Computation*, 2nd ed. Baltimore: John Hopkins University Press, 1989

Two-Dimensional Robot Navigation Among Unknown Stationary Polygonal Obstacles

Guy Foux, Michael Heymann, and Alfred Bruckstein

***Abstract*—We describe an algorithm for navigating a polygonal robot, capable of translational motion, in an unknown environment. The environment contains stationary polygonal obstacles and is bounded by polygonal walls, all of which are initially unknown to the robot. The environment is learned during the navigation process by use of a laser range-finding device, and new knowledge is integrated with previously acquired information. A partial map of the environment is thus obtained. The map contains parts of the obstacles that were "seen" by the robot and the free space between them. The obstacles in the map are transformed into a new set of expanded polygonal obstacles. This enables treating the robot as a point instead of a polygon, and the navigation problem is reduced to point navigation among unknown polygonal obstacles. A navigation graph is built from the transformed obstacles in the map. This is a partial visibility graph of the enlarged obstacles. A search is conducted on the graph for a path to the destination. The path is piecewise linear; at its corners, the robot stops, scans its environment, and updates the map, the obstacles, and the planned path. The algorithm is proved to converge to the desired destination in a finite number of steps provided a path to the destination exists. If such a path does not exist, then the navigation process terminates in a finite number of steps with the conclusion that the destination is unreachable.**

I. Introduction

The problem of robot navigation in an unknown environment can be described as follows: source and destination points are given, the robot being outside all obstacles when placed on any of these points. The navigation space is unknown and may contain obstacles of different kinds. The problem is to find a path from the source point to the destination point, so that motion of the robot along this path is such that the robot is safe from collision with the obstacles. To implement the navigation, the robot uses information on the environment provied by its sensors.

An algorithm for solving the navigation problem for a polygonal robot in a two-dimensional unknown environment, where the obstacles are stationary polygons, is presented in this paper. The assumptions made in this work are that there are a finite number of stationary polygonal obstacles with a finite number of vertices, and that the robot polygon also has only a finite number of vertices. The robot is assumed to sense its environment with a range sensor providing the distance to the first obstacle in all directions. This is a good model for a laser-sensing device, and the range and angular sensing, as well as the motion, are assumed to be error free. The algorithm has the following properties:

1) *Convergence*—If a path from source to destination exists, then the robot reaches the destination in a finite number of steps. If such a path does not exit, then, in a finite number of steps, the robot reaches this conclusion and halts.
2) *Learning*—The robot learns its envirnoment during the navigation process. A map representing this knowledge (i.e., obstacle walls and free space) is kept and updated and is used for planning the navigation path.

Manuscript received January 10, 1991; revised June 12, 1992. M. Heymann and A. Bruckstein were supported in part by the Technion Fund for the Promotion of Research

The authors are with the Center for Intelligent Systems, Department of Computer Science, Technion, I.I.T., Haifa, Israel.

IEEE Log Number 9205074.

3) *Monotonic Behavior*—The length of paths, produced by the navigation algorithm, is a monotonic nonincreasing function of the available knowledge (which is monotonically accumulated). If knowledge of the environment were complete (i.e., if all the obstacles and the free space between them were known), the paths produced would be otimal in the sense of the shortest Euclidean length.
4) *Environment Complexity*—There are no further limitations on the shape of the obstacles, like convexity, and therefore the algorithm solves navigation problems in complex environments such as mazes.
5) *Polynomial Time Complexity*—The complexity of calculations for any navigation step are third-order polynomials of the number of vertices in the obstacle polygons. The complexity of calculations for the whole navigation process is a fourth-order polynomial of the number of the vertices.

In recent years, the navigation problem in unknown environments was often addressed in the literature. However, for the set of assumptions made in the present paper, no solution including all of the above properties has been published. The algorithms proposed by Cahn and Phillips [3], Koch *et al.* [9], Moravec [17], and Thompson [24] are not convergent. The algorithms presented by Lumelsky and Stepanov [13], [14] and Lumelsky [15] are convergent. Learning, however, is not incorporated, and therefore there are no improvements in the performance, even if a specific task is repeated over and over again. On the other hand, these algorithms can solve navigation problems in very complex situations such as nonpolygonal mazes. Recently, Lumelsky *et al.* [16] addressed learning and terrain model acquisition within the framework of their navigation scheme. The algorithm proposed by Iyengar *et al.* [8] was not formally proven to converge. This algorithm employs learning of the environment and there are improvements in the planned paths with the accumulation of knowledge. However, there tend to be unnecessary detours in the paths due to the navigation strategy. The algorithm by Oommen *et al.* [19] works for a point robot in environments where the obstacles are convex polygons but does not necessarily converge in all situations. Rao and Iyengar [21] and Rao *et al.* [20] described a convergent algorithm that also learns the environment. Paths are generated by a combination of local and global strategies. This involves definition of subgoals, which again tend to yield unnecessary detours in the overall paths to the destination. More recently, Rao *et al.* [22] described an interesting algorithmic approach based on retraction in which navigation is implemented along the Voronoi diagram of the terrain.

The solution presented in the following sections is a navigation scheme for a polygonal robot capable of translational motion only. In order to reduce the problem of navigating a polygon to that of navigating a point, the obstacles are enlarged by the robot polygon's dimensions to yield a new set of polygonal obstacles. This "enlargement" of the obstacles is a well known method introduced formally by Lozano-Perez and Wesley [11].

II. Problem Definition

The robot is a polygon in a polygonally bounded region B in the plane, such that the boundary of B has a finite number of vertices. Inside region B, there are a finite number of stationary polygonal obstacle with a finite number of vertices, such that each vertex is the intersection of at most two edges. Therefore, an obstacle can have a closed polygonal boundary, or it can be an open polygonal "wall" (see Fig. 1).

Let $W = \{w_1, w_2, \cdots, w_N\}$ be the set of all obstacles in B. We shall denote the boundary of an obstacle $w_m \in W$ by bd(w_m) and its interior by int(w_m) (for every open polygonal wall int$(w_m) = \emptyset$).

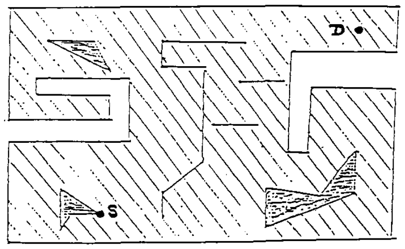

Fig. 1. Domain B with the obstacles, some of which are polygonal bodies while others are polygonal walls. Points S and D are the source and destination points, respectively.

In the same manner, bd(B) and int(B) denote the boundary and interior of the region B, repsectively. We assume that every obstacle is contained in int(B) (i.e., does not intersect bd(B)) and that no two obstacles intersect.

Let FS denote the feasible free space, which is the set of all points in B in which the robot's reference point may be placed without causing collision between the robot and any of the obstacles $w_m \in W$. In FS there is a point S in which the reference point of the robot is initially situated, and a point D, which the robot aims to reach. The robot can move along straight lines only, and therefore a feasible path from S to D is a piecewise linear path in FS. When the robot's reference point moves along such a path, the robot is outside all obstacles in B. For the sake of simplicity of the navigation principles, we assume this reference point to be in the interior of the robot's polygon and not on its boundary.

The "navigation problem" is to find a feasible path from S to D. The solution to this problem, by finding a path and moving along it, is called a "navigation task." The motion starts at the initial position $S_1 = S$ and proceeds through intermediate points S_i $(i = 2, 3, \cdots)$, which are vertices of a piecewise-linear path, to the destination point D or to a point S_k, where it becomes evident that the destination is unreachable. At the points S_i, the robot performs a range-sensing sweep in all directions, giving in each direction the distance to the nearest obstacle. This process supplies the robot with increasing amounts of information on its environment, thereby enabling it to make decisions concerning the next move.

III. Navigation Principles

A. The Learned Free Space

We assume that the robot's reference point is situated at a point S_k on the navigation path. The parts of the free space "seen" by the robot from S_k, and from some specific points along the path that lead to S_k, are called the *learned free space* corresponding to S_k and marked LFS_k. The next section describes how this LFS_k is built and updated, but for the time being it is enough to know that LFS_k is that part of the free space with which the robot is familiar (through learning).

The boundary of LFS_k is divided into a finite number of *learned boundaries* E_j. These are connected parts of obstacle boundaries that were seen by the robot. Each learned boundary $E_j \in LFS_k$ is either contained in the boundary of B or in the boundary of some obstacle $w_m \in W$ in B. The view of some of the learned boundaries might be obstructed by other learned boundaries. In such a case, the line segment connecting the endpoint of the obstructed obstacle with the corresponding endpoint of the obstructing obstacle is called a *temporary segment.*

It is assumed, for sake of planning a path, that, at each stage, the learned boundaries known to the robot constitute the complete set of

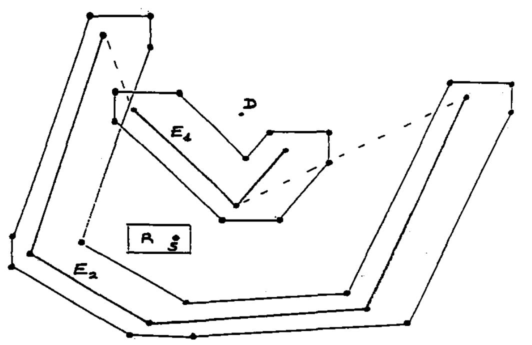

Fig. 2. The configuration obstacles. R is the robot, and the broken lines represent the temporary segments that bound LFS_k.

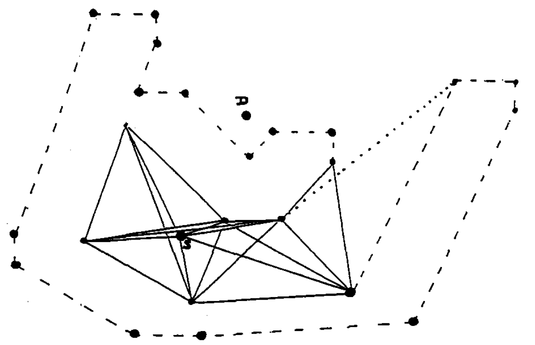

Fig. 3. Visibility lines between vertices of the configuration obstacles. The full lines are Type A visibility lines, the broken lines are Type B visibility lines, and the dotted ones are Type C visibility lines.

obstacles in B. This implies that the entire space between and around them is free space. The assumption is generally false, of course, but it enables planning a path in unknown regions.

B. The Alleged Feasible Free Space

In order to treat the robot as a point, the obstacles known to the robot, namely, the learned boundaries, are grown by the robot's polygonal dimensions. A technique for "growing" obstacles was presented in 1979 by Lozano-Perez and Wesley [11], and treated formally in 1983 by Lozano-Perez [12]. Growing a polygonal obstacle by another polygonal object (the robot) that can only perform translational motion yields a new obstacle that is also a polygon. See Fig. 2. We use the term *configuration obstacle* to describe the grown obstacle because the original obstacle was transformed into the configuration space of the robot's reference point (which in this case is the (X, Y) space of the location of the reference point in the plane). If two configuration obstacles intersect, then their union is treated as a single configuration obstacle. Each vertex in the configuration obstacle is *related* to exactly one vertex of the original learned boundary.

The space between the configuration obstacles is considered to be safe for motion (based on the assumption of the previous subsection). This alleged free space is marked AFS_k for *alleged feasible free space*. The problem is to move the reference point from its current position to a new position in AFS_k in a manner that will eventually bring the reference point to the destination.

C. The Navigation Graph

1) Visibility Lines: A visibility line between two vertices V and U of the configuration obstacles is the straight line segment (V, U) that belongs to one of the following categories (see Fig. 3):

1) A visibility line that is contained in LFS_k: In this case the visibility line between the vertices V and U is a true visibility line, because it passes in a region known to be free of obstacles.
2) A visibility line along an edge (V, U) of a configuration obstacle not contained in LFS_k: In this case the visibility line between the vertices V and U is a *plausible visibility line* because it relies on the assumption that the space around the configuration obstacles is free.
3) A visibility line corresponding to a temporary segment: This case occurs if the vertices related to V and U, in the original learned boundary, are both endpoints of their appropriate learned boundaries, between which a temporary segment exists (i.e., an obstruction occurred). A *plausible visibility line* exists between any of the pairs of $\hat{V}$ and $\hat{U}$ related to V and U in the configuration obstacles (unless there are $\hat{V}$ and $\hat{U}$ that fit in the first category).

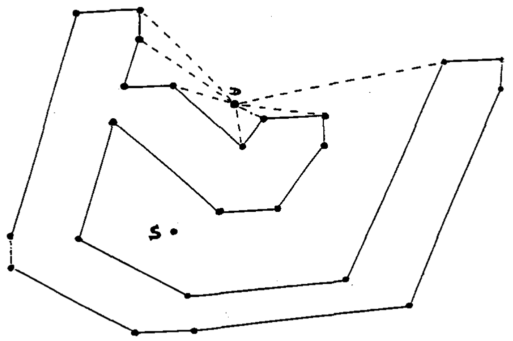

Fig. 4. Plausible visibility lines from D.

Comment: Since $S_k \in LFS_k$, we consider S_k to be a vertex for the purpose of applying the definition of visibility lines to S_k.

We now define visibility lines and plausible visibility lines of the destination point D (see Fig. 4).

A *visibility line from D* is the straight line segment from D, totally contained in LFS_k, to a vertex V in LFS_k. Such visibility lines exist only if $D \in LFS_k$.

A *plausible visibility line from D* is the straight line segment from D to a vertex V in LFS_k that does not intersect any edge of a configuration obstacle (except for its endpoints D and V), and at least part of it is outside LFS_k.

2) Definition of the Navigation Graph: The navigation graph NG_k (corresponding to S_k) is the following directed graph:

1) A single node in the graph corresponds to the destination point D and to each of the vertices from the configuration obstacles.
2) A single node in the graph corresponds to the robot's position S_k, if S_k is not a vertex in any of the configuration obstacles.
3) Two antiparallel directed arcs $(\overrightarrow{V, U})$ and $(\overrightarrow{U, V})$ correspond in the graph to every visibility line or plausible visibility line, between vertices V and U (including S_k).
4) A single directed arc $(\overrightarrow{V, D})$ corresponds in the graph to every visibility line or plausible visibility line (D, V) from D.
5) A cost is assigned to every arc in the graph that corresponds to a true visibility line between vertices or a visibility line from D. The cost of each such arc is equal to the Euclidean distance between the vertices corresponding to the nodes at its ends.
6) A cost is assigned to every arc in the graph that corresponds to a

plausible visibility line between vertices or a plausible visibility line from D. The cost of each such arc is equal to the product of a constant conservation/curiosity factor (CCF), set by the user, and the Euclidean distance between the vertices corresponding to the nodes at its ends. The constant CCF determines the will of the robot to navigate in unmapped regions. A large value for CCF encourages conservative navigation in the known regions, whereas a small value for CCF encourages more adventurous navigation.

The navigation graph NG_k is a finite directed graph whose weights are non-negative, and therefore Dijkstra's algorithm [5], for finding minimal paths in graphs, is applicable.

If at a point S_k there is an increase in knowledge ($LFS_k \neq LFS_{k-1}$), then the navigation graph is updated. If there is no increase in knowledge, then the only change in NG_k relative to NG_{k-1} is marking a new node as corresponding to S_k, and removing the note corresponding to S_{k-1} if S_{k-1} is not a vertex of any configuration obstacle in AFS_k.

3) Existence of a Path to Destination D in the Navigation Graph:

Lemma 1: If a path from S to D exists in B, then a path exists in NG_k from S_k (the node corresponding to the robot position) to the node corresponding to the destination D.

Proof: Let us consider the two-dimensional free-space AFS_k between the grown learned boundaries of LFS_k. Since every learned boundary $E_j \in LFS_k$ is contained in the boundary of B (bd(B)) or the boundary of some obstacle $w_m \in W$ in B (bd(w_m)), it follows that the real feasible free space FS, between the configuration obstacles in B, is contained in the alleged feasible free space, $FS \subset AFS_k$. If a path from S to D exists in FS, then S and D belong to a connected subspace of FS, and therefore to a connected subspace of AFS_k, which will be denoted $\overline{AFS_k}$.

D "sees" some vertices of configuration obstacles in AFS_k, and corresponding arcs exist in NG_k. To each of these vertices a path from S_k exists in the navigation graph, because $\overline{AFS_k}$ is a connected region that includes both S and S_k. Therefore, a path exists in the navigation graph between the nodes corresponding to S_k and to D.■

Hence, if no path from S_k to D exists in the navigation graph NG_k, then no path exists from S to D in the domain B. From this conclusion we draw the *termination condition* of the algorithm: If at some point S_k on the navigation path the navigation graph NG_k contains no paths from S_k to D, then the robot halts at S_k and the navigation task terminates with the conclusion that the destination is unreachable.

Note that in the case where no path exists from S to D, the robot might move from $S_1 = S$ to some S_k ($k > 1$) before the process terminates. In NG_k there is no path from S_k to D. However a path to D did exist in the navigation graphs corresponding to all the points $S_1, S_2, \cdots, S_{k-1}$.

D. The Navigation Algorithm

Planning and executing motion is conceptually composed of the following three steps:

Step 1: Suppose the robot's reference point is situated at point S_k ($k \geq 1$), the robot scans its surroundings to obtain the "seen" part of the environment from S_k, and updates the learned free space LFS_k accordingly.

Step 2: If $LFS_k \neq LFS_{k-1}$, then the navigation graph is updated (for $k = 1$ the graph is built), and a new path to the destination D is planned by applying Dijkstra's algorithm to NG_k. If no path exists in NG_k from S_k to D, then the algorithm terminates with the conclusion that the destination is unreachable. If a path exists, then let the planned path be $s_k \rightarrow V_i \rightarrow V_{i+1} \rightarrow \cdots \rightarrow D$. The robot moves from S_k to V_i along a straight line. If $V_i = D$, then the process terminates successfully. If $V_i \neq D$, then V_i is marked S_{k+1}, and step 1 is performed again.

Step 3: If $LFS_k = LFS_{k-1}$, then the robot continues with the planned path and moves along a straight line from S_k to the next vertex on the path V_j. If $V_j = D$, then the algorithm terminates successfully. If $V_j \neq D$, then V_j is marked S_{k+1} and step 1 is performed again.

The robot can only move along visibility lines from S_k to some vertex $V \in AFS_k$. Analysis of the visibility lines from S_k reveals all of them to be true visibility lines (and not plausible visibility lines), and therefore all these visibility lines are contained in LFS_k. V is therefore related to a permanent vertex in LFS_k. (Had V been related to a temporary vertex, obtained through obstruction, then V would have been obstructed by the grown part of the original obstructing obstacle.)

E. Convergence of the Navigation Algorithm

Theorem 1: If a path from S to D exists, then the algorithm converges to the destination point D in a finite number of steps. If such a path does not exist, then the algorithm terminates in a finite number of steps with the conclusion that the destination is unreachable.

Proof: If there is a planned path from $S_1 = S$ to D, then the robot moves along the path until D is reached or until a change occurs in LFS. If there is no such initial path, then there is no path to D (Lemma 1) and the algorithm terminates (termination requirement). After every step, when the robot reaches S_k, if $S_k = D$, then the navigation terminates successfully. If $S_k \neq D$, then the updated LFS_k is compared with LFS_{k-1}. If a change has occurred in LFS_k, then the set of configuration obstacles and the navigation graph are updated, and a new path to D is planned. If no such path exists in the graph, then the destination is unreachable (Lemma 1) and the algorithm terminates. If there is a planned path, then the robot continues its motion along the path until destination D is reached or another change occurs in LFS.

Changes in LFS take place only at points that the robot visits for the first time because of the stationarity assumption. At each step, the robot moves from S_k to a vertex V related to a permanent vertex in LFS_k. The number of permanent vertices in LFS_k is finite, and therefore the number of vertices, in the configuration obstacles, related to them is also finite. Therefore, the number of changes in LFS is bounded. After the last change, there either is a path to D (that will not change again), or there is no path at all. In the first case, D is reached and, in the latter, the algorithm terminates with the conclusion that D is unreachable. ■

Note that the proof relies on the finite number of points in B to which the robot can move.

IV. Learning

In this section, we discuss the methods by which knowledge is acquired and stored. The robot needs a sensing device that would enable it to sense and learn the world around it. We therefore assume that the robot is equipped with a laser range-finding device, capable of measuring the exact distance to the nearest obstacle, or to the boundary of B, in any direction θ.

A. The Free Zone

By performing an angular laser scan of 360° from the robot's reference point, which is positioned at S_i, the "seen" part of the environment from S_i is obtained. See Fig. 5. The boundary of this seen part can be represented as a single valued function $r(\theta)$, defined

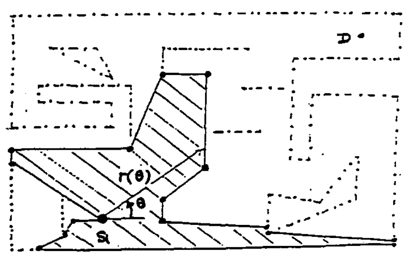

Fig. 5. The "seen" part of the environment from S_i. Each point on the boundary of the seen part has an $r(\theta)$ representation.

on the interval $[0, 2\pi)$. θ is the angle that a ray from S_i forms with a predefined reference direction in the plane, and $r(\theta)$ is the distance from S_i to the nearest point on that ray that is also on the boundary of B or of an obstacle in B. The function $r(\theta)$ is defined for all θ since B has a closed boundary.

The function $r(\theta)$ has a finite number of discontinuity points, because discontinuities occur only at angles where there is an obstruction of one obstacle by another. Between every two consecutive discontinuity points, $r(\theta)$ describes part of a boundary of some obstacle which is called a *learned boundary*. The learned boundary is composed of straight line segments called *learned edges*. The endpoints of the learned edges are called *vertices*. The vertices are divided into two categories: 1) *permanent vertices*, which are closed ends of learned edges, and 2) *temporary vertices*, which are open ends of learned edges, created due to an obstruction by another obstacle (see Fig. 6).

At each discontinuity point we have a transition from a learned edge with a "near" closed end to a learned edge with a "far" open end. In each discontinuity point, the temporary vertex is connected to the permanent vertex that has created it by a straight line segment called a *temporary segment*. Adding these temporary segments to the seen part of the environment from S_i creates a region of free space that is called the *free zone* from S_i and is marked FZ_i. The boundary of FZ_i is a closed polygon with a finite number of edges. This region is a star-shaped object, with S_i located in the kernel, and therefore it is a connected region.

A point V in FZ_i is chosen, as we explained in the previous section, and the robot moves to it. If this point is the destination point D, then the navigation task terminates successfully. Otherwise, the robot stops at V, marks it as S_{i+1}, and scans the environment from S_{i+1} to obtain FZ_{i+1}.

B. The Learned Free Space

The LSF is defined as follows:

$$LFS_k = \bigcup_{i=1}^{k} FZ_i = \begin{cases} FZ_1, & k = 1 \\ LFS_{k-1} \cup FZ_k, & k > 1 \end{cases}. \quad (1)$$

Since LFS_k is a finite union of free zones, where each free zone is a polygonal region whose boundary has a finite number of edges, then LFS_k is also a polygonal region in the plane whose boundary consists of a finite number of straight line segments. These line segments are either learned edges or temporary segments, as depicted in Fig. 6. LFS_k is a connected region since it is a union of free zones, each of which is a connected region, such that every two consecutive free zones have a common point.

The endpoints of all learned edges in LFS_k are called *vertices*, and therefore the number of vertices in LFS_k is finite. A vertex is called a *permanent vertex* if it was a permanent vertex in either LFS_{k-1} or FZ_k. Otherwise, the vertex is called *temporary*.

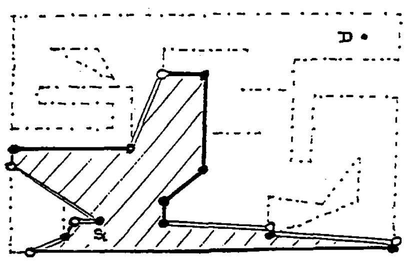

Fig. 6. The free zone FZ_i. The full line segments are "learned edges" while the hollow ones are "temporary segments." The endpoints of the line segments are the vertices, the solid dots are permanent vertices, and the hollow dots are temporary vertices.

The boundary of LFS_k is divided into a finite number of *learned boundaries* E_j, which are connected groups of learned edges that were seen by the robot. Each learned boundary in LFS_k is contained in the boundary of B or in the boundary of some obstacle $w_m \in W$ in B.

The following lemma is a straightforward observation and is therefore presented here without proof. The interested reader will find the proof in [6] (see also [7]).

Lemma 2: Every permanent vertex in LFS_k is a vertex of some obstacle $w_m \in W$ or of bd(B).

We say that there is an *increase in knowledge* of the envirnoment if $LFS_k \neq LFS_{k-1}$. If at some stage of the navigation process $S_k = S_i$, where $i = 1, 2, \cdots, k-1$ (i.e., a second visit to the same point), then it is clear from the stationarity assumption that $LFS_k = LFS_{k-1}$ and there is no increase in the robot's knowledge of the environment.

C. Additional Navigation Tasks

When a navigation task terminates, either upon reaching the destination or by stopping the navigation process (when the destination is unreachable), the free space learned while navigating is marked as LFS^J $(J = 1, 2, \cdots)$, for possible use in future navigation tasks.

If a new navigation task from point S to point D is requested, then at every step k (corresponding to a robot position at S_k) it is checked whether FZ_k has a common point with and LFS^J obtained in a previous navigation task in B. If no such point is detected, then the navigation process continues as if this was the first navigation task in B. In the case where a common point is found, the LFS_k from the current task is united with the LFS^J from the previous task. The current navigation process continues using this updated LFS_k as its knowledge base.

V. Discussion

A. Complexity Analysis

Let us begin by analyzing the complexity of a single step of the algorithm, which is the complexity of calculating S_{k+1} when the robot is at S_k.

Learning involves integrating new knownedge FZ_k, with previous information LFS_{k-1}. Each line segment in the boundary of FZ_k must be checked for intersection against all line segments in LFS_{k-1} to obtain the line segments of LFS_k. If in LFS_{k-1} are N_{k-1} vertices, and in FZ_k are n_k vertices, then the complexity of this calcuation is $O(n_k \times N_{k-1})$.

Calculation of the configuration obstacles corresponding to the learned boundaries in LFS_k is of $O(N_k \times N_R)$, where N_k is the number of vertices in LFS_k, and N_R is the number of edges in the robot's polygon. The configuration obstacles obtained have $O(N_k \times N_R)$ vertices.

For determining the visibility connections inside AFS_k, the line segment between every two vertices of the configuration obstacles (i.e., the proposed visibility line) must be checked for intersection against all line segments in the boundary of LFS_k. The number of vertices in AFS_k is $O(N_k \times N_R)$, and therefore the complexity of this calculation is bounded by $O(N_k^3 \times N_R)$. Visibility lines along the edges of the configuration obstacles are calculated in $O(N_k \times N_R)$. Finding the vertices "seen" from the destination point D involves using a plane-sweeping technique such as the one described by Sharir and Shorr [23], whose complexity is $O((N_k \times N_R)^2 \times \log(N_k \times N_R))$. Therefore, the complexity of building the navigation graph NG_k from AFS_k is bounded by $O((N_k \times N_R)^3)$.

Searching the graph for a path is done by using Dijkstra's algorithm, whose complexity is $O(V \times E)$, where V is the number of nodes, which is $O(N_k \times N_R)$, and E is the number of arcs in the graph bounded by V^2. The search complexity is, therefore, bounded by $O((N_k \times N_R)^3)$.

We have thus established that the time complexity for a single step is bounded by $O((N_k \times N_R)^3)$, where N_k is the number of vertices in LFS_k, and N_R is the number of edges in the robot's polygon.

The number of vertices N_k in LFS_k is $O(N)$, where N is the number of vertices in B. The intermediate goals can be vertices of the configuration obstacles, related to permanent vertices in LFS_k, and therefore there are $O(N \times N_R)$ points that can serve as intermediate goals for the robot's motion. Therefore, the time complexity of the whole navigation process is bounded by $O((N \times N_R)^4)$.

B. Monotonic Behavior

If the robot had full knowledge of its environment, then the navigation graph would be the full visibility graph of the configuration obstacles. Dijkstra's algorithm would then find the least expensive path between the vertices. Based upon total knowledge of the environment, this path is optimal in the sense of minimization of the Euclidean distance along the path.

The learning process introduced here is monotonic since knowledge can only increase. Therefore, paths can only improve with the increase of knowledge and would be optimal if knowledge of the environment were complete.

VI. Summary and Conclusion

An algorithm for navigating a polygonal robot, capable of translational motion, in an unknown environment with polygonal obstacles was presented. The algorithm plans and executes a piecewise-linear path between the source and destination points. The algorithm was shown to converge to the destination in a finite (and bounded) number of steps if the destination is reachable or to terminate in a finite number of steps if the destination is unreachable. The algorithm has also been shown to operate and converge (with minor adjustments) for the special cases of a point robot and a disk (two special cases of deteriorated polygons). See [6], [7].

Throughout the navigation, the robot maps and learns its environment by performing laser scans at the corners of its piecewise linear path and integrating new information with the existing knowledge. A map depicting the free space seen by the robot from the corners of the path, and the walls that bound it, is thus obtained. This map is used for planning the navigation path and is kept for additional navigation tasks in the same envirnoment in the future.

The algorithm solves navigation problems in very complex environments such as polygonal mazes. This is because the only assumptions made are that the obstacles are stationary polygons with a finite number of vertices.

The time complexity of the process was shown to be polynomial in the number of vertices of the obstacles in B and its boundary. (A third-order polynomial for each step and a fourth-order for the whole process.) This complexity enables practical use of the algorithm for navigation in real environments.

We have assumed throughout this paper that the robot performs exact motion. This assumption is not practical because of the phenomenon of wheel slippage, which creates errors between the planned and actual path executed by the robot. This assumption can be omitted if a feedback loop is introduced for motion control using the range-finding device so as to ensure that the desired path is accurately executed. An alternative approach [4] is to use the map created by the robot for updating the estimation of the robot's position. The update is achieved by comparing the picture obtained from the current range scan with the map. In order to match the two maps, orientation and translation corrections are performed to estimate the robot's position. After the estimation of the robot's position is corrected, the map is updated in the manner described earlier.

The problem of inexact range readings (as obtained with sonar range finders) is a much more complex problem that has received a lot of attention lately. Among the papers that have dealt with this problem are Ayache *et al.* [1] and Kriegman *et al.* [10] who have represented the uncertainty as a normally distributed function and used the extended Kalman filter to minimize it. Moravec [18] used sensor redundancy and averaging in order to minimize uncertainty. Brooks [2] used a relational map that represented relationships between parts of the world, with their associated uncertainties, rather than trying to build a map of the world in a fixed coordinate system.

Acknowledgment

The authors would like to thank the referees for their constructive comments and suggestions.

References

[1] N. Ayache and O. D. Faugeras, "Maintaining representations of the environment of a mobile robot," *IEEE Trans. Robotics Automat.*, vol. 5, no. 6, pp. 804–819, Dec. 1989.

[2] R. A. Brooks, "Visual map making for a mobile robot," in *Proc. IEEE Int. Conf. Robotic Automat.*, 1985, pp. 824–829.

[3] D. F. Cahn and S. R. Phillips, "ROBNAV—A range-based robot navigation and obstacle avoidance algorithm," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-5, no. 5, pp. 544–551, Sept. 1975.

[4] J. Crowley, "Navigation for an intelligence mobile robot," *IEEE J. Robotics Automat.*, vol. RA-1, no. 31, pp. 31–41, Mar. 1985.

[5] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, pp. 269–271, 1959.

[6] G. Foux, M. Heymann, and A. Bruckstein, "Two-dimensional robot navigation among unknown stationary polygonal obstacles," Technion, IIT, Haifa, Israel, CIS Rep. 9001, Dec. 1990.

[7] G. Foux, "Two-dimensional disk-robot navigation in an unknown environment among polygonal obstacles," M.Sc. thesis, Technion-Israel Institute of Technology, Haifa, Israel, Feb. 1990 (in Hebrew).

[8] S. S. Iyengar, C. C. Jorgensen, S. V. N. Rao, and C. R. Weisbin, "Robot navigation algorithm using learned spatial graphs," *Robotica*, vol. 4, pp. 93–100, 1986.

[9] E. Koch, C. Yeh, G. Hillel, A. Meystel, and C. Isik, "Simulation of path planning for a system with vision and map updating," in *Proc. IEEE Int. Conf. Robotics Automat.*, vol. 1, pp. 146–160, Mar. 1985.

[10] D. J. Kriegman, E. Triendl, and T. O. Binford, "Stereo vision and navigation in buildings for mobile robots," *IEEE Trans. Robotics Automat.*, vol. 5, no. 6, pp. 792–803, Dec. 1989.

[11] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision free paths among polyhedral obstacles," *Communication*, vol. ACM-22, no. 10, pp. 560–570, Oct. 1979.

[12] T. Lozano-Perez, "Spatial planning—A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108–120, Feb. 1983.

[13] V. J. Lumelsky and A. A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE trans. Automat. Contr.*, vol. AC-31, no. 11, pp. 1058–1063, Nov. 1986.

[14] ——, "Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, vol. 2, pp. 403–430, Nov. 1986.
[15] V. J. Lumelsky, "Dynamic path planning for a planar articulated robot arm moving amidst unknown obstacles," *Automatica*, vol. 23, no. 5, pp. 551–570, 1987.
[16] V. J. Lumelsky, S. Mukhopadhyay, and Kang Sun, "Sensor-based terrain acquisition: The "Sightseer" strategy," in *Proc. 28th IEEE Conf. Decision Contr.*, (Tampa, FL), Dec. 1989, pp. 1157–1161.
[17] H. P. Moravec, "Rover visual obstacle avoidance," in *Proc. 7th IJCAI* (Vancouver, B.C., Canada) Aug. 1981, pp. 785–790.
[18] ——, "Visual mapping by a robot rover," in *Proc. 6th IJCAI* (Tokyo), Aug. 1979, pp. 589–600.
[19] B. J. Oommen, S. S. Iyengar, N. S. V. Rao, and R. L. Kashyap, "Robot navigation in unknown terrains using learned visibility graphs, Part 1: The disjoint convex obstacle case," *IEEE Trans. Robotics Automat.*, vol. RA-3, no. 6, pp. 672–681, Dec. 1987.
[20] N. S. V. Rao, S. S. Iyengar, and G. deSaussure, "The visit problem: Visibility graph based solution," in *Proc. IEEE Int. Con. Robotics Automat.*, vol. 3, 1988, pp. 1650–1655.
[21] N. S. V. Rao and S. S. Iyengar, "Autonomous robot navigation in unknown terrains: Incidental learning and environmental exploration," *IEEE Trans. Syst. Man Cybern.*, vol. 20, pp. 1443–1449, 1990.
[22] N. S. V. Rao, N. Stoltzfus, and S. S. Iyengar, "A retraction method for learning navigation in unknown terrains for a circular robot," *IEEE Trans. Robotics Automat.*, vol. 7, pp. 699–707, 1991.
[23] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," *Siam J. Comput.*, vol. 15, no. 1, pp. 193–215, Feb. 1986.
[24] A. M. Thompson "The navigation system of the JPL robot," in *Proc. 5th IJCAI* (MIT, Cambridge, MA) Aug. 1977, p. 749–757.

High-Speed Trajectory Control of a Direct-Drive Manipulator

K. Youcef-Toumi and A. T. Y. Kuo

***Abstract*—The trimming of three-dimensional parts using laser-cutting industrial robots raises a control challenge when high speeds and precision are required. Accurate control of robot movement along predetermined trajectories is necessary in order to achieve satisfactory cuts. This paper focuses on the control system design for direct-drive manipulators specially designed for high-speed trajectory control applications. First, the concept of decoupled and invariant dynamics is discussed for a specific manipulator. Second, a simple procedure for system identification and control system design is presented. It is demonstrated that, through arm mechanism design, the control system is greatly simplified and satisfactory control performance is achieved. The arm mechanism design and control system are evaluated through simulations and experiments. The experimental tracking performance achieved is characterized by a speed of 3 m/s and an acceleration of 3.8 g, with a joint mean tracking error of 0.0556°.**

I. Introduction

Motion planning of industrial robots has evolved from simple point-to-point playback of the end-effector to complex trajectory following. Spot welding and arc welding by industrial robots differ in that the latter application requires the end-effector to follow a desired trajectory in space. Thus, arc welding requires amore complex trajectory planning. A greater challenge has been raised recently in the application of laser cutting to sheet metal. This process requires both high-speed maneuvering of the end-effector and accurate tracking. Specifically, the speed, acceleration, and tracking accuracy required are on the order of 1–3 m/s, 3–5 g, and 0.05–0.1 mm, respectively.

Manuscript received December 10, 1987; revised January 18, 1992. This work was supported by the National Science Foundation and by ShinMaywa Co. of Japan.

K. Youcef-Toumi is with the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139.

A. T. Y. Kuo was with the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139. He is now with the Tuntex Group, Tuntex Petrochemicals, Inc., Taiwan.

IEEE Log Number 9205079.

The difficulties in performing high-speed trajectory tracking with conventional robots are numerous. Limitations on the speed and accuracy of the robots are imposed by the drive system's components, such as gearing, lead screws, and linkage, because of its compliance. In order to overcome these difficulties, direct-drive robots were introduced [2]–[4], [6]. By removing the transmission systems, the backlash, friction, and compliance of the drive system have been eliminated. In addition, advanced composite materials were used in the linkage of the high-speed M.I.T. direct-drive arm. Consequently, the arm linkage stiffness was increased significantly and the arm inertia reduced [6]. Therefore, the control of this direct-drive robot at the joint ensures fast and accurate tracking of the endpoint in task space.

A few papers have been published in the area of trajectory tracking of direct-drive robots [1], [2], [7]. All of the results were obtained on direct-drive robots with open kinematic chain structures. These structures exhibit significant coupling and interactions between the different joints. Nevertheless, the main control algorithm used by the researchers is based on a feedforward action that can be effective. The first experimental results for the direct-drive concept [2] showed promise. Maximum joint speeds ranged from 180 to 360°/s. Positioning accuracy measurements were also conducted using step responses. This was accomplished by commanding the direct-drive robot to move to a target point several times. The measured accuracy was -0.287. Experimental results published recently [1], [7] were also obtained using model-based feedforward controls. The performance of model-based feedforward controllers depends greatly on model accuracy. The feedforward model usually consists of robot dynamic equations used to calculate the torques/forces necessary to drive the robot along the desired trajectory. These equations are highly nonlinear and are functions of robot parameters. The parameters of the model include link inertial parameters, actuator characteristics, and other relevant system parameters. In [7], the link inertial parameters were estimated from detailed drawings of a geometric solid model of the robot. The approach adopted in [1] is to estimate the model inertial parameters through arm excitation. These approaches can provide satisfactory results when appropriate algorithms and adequate computing hardware are used. Reference [10] describes the control of a two-degree-of-freedom (2-DOF) decoupled parallel direct-drive arm using preview control.

One of the major reasons in using feedforward control is to compensate for interactions between joints caused by nonlinear effects such as centrifugal and Coriolis forces. The published results mentioned above are for open kinematic chain manipulators, which are dynamically coupled and nonlinear.

An alternative approach to achieving satisfactory tracking performance is to consider both the robot arm mechanism design and the controller design. The M.I.T. direct-drive arm for laser cutting applications was designed with these issues in mind. Through appropriate design and mass redistribution techniques, the arm dynamics is made decoupled and inertia invariant [4]–[6], [11]. The expressions