A Parallel Algorithm for Computing 3-D Reachable Workspaces

Tarek Alameldin and Tarek Sobh

Department of Computer Science
California State University at Fresno
Fresno, CA 93740-0109

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania, Philadelphia, PA 19104

Abstract

The problem of computing the 3-D workspace for redundant articulated chains has applications in a variety of fields such as robotics, computer aided design, and computer graphics. The computational complexity of the workspace problem is at least NP-hard. The recent advent of parallel computers has made practical solutions for the workspace problem possible. Parallel algorithms for computing the 3-D workspace for redundant articulated chains with joint limits are presented. The first phase of these algorithms computes workspace points in parallel. The second phase uses workspace points that are computed in the first phase and fits a 3-D surface around the volume that encompasses the workspace points. The second phase also maps the 3-D points into slices, uses region filling to detect the holes and voids in the workspace, extracts the workspace boundary points by testing the neighboring cells, and tiles the consecutive contours with triangles. The proposed algorithms are efficient for computing the 3-D reachable workspace for articulated linkages, not only those with redundant degrees of freedom but also those with joint limits.

1 Introduction

The problem of computing three dimensional workspaces for redundant articulated chains has applications in a variety of fields such as robotics, computer-aided design, and computer graphics. The reachable workspace of an articulated chain is the volume or space encompassing all points that a reference point P on the hand (or the end effector) traces as all the joints move through their respective ranges of motion [13]. An articulated chain is a series of links connected with either revolute or prismatic joints such as a robotic manipulator or a human limb.

In computer-aided design, the three dimensional workspace of a human limb or robotic manipulator can be used in the design of the interior parts of cars, tanks, or space vehicles. Different panels and keys can be repositioned within the 3-D reachable workspace. These different configurations can be tested before being manufactured.

Three dimensional workspaces can also be used in the interfacing between computer graphics and artificial intelligence. Task plans for approaching and grasping an object in either a static or a

 0-8194-0873-5/92/$4.00

dynamic environment can be formulated by computing the reach capabilities of different agents[1] and passing this information, along with the current geometry of the environment, to the planner. The planner then decides the best way to approach and grasp the object, and also decide which agent is best to use.
The workspace problem has long been on the agenda of robotics researchers [12,21,10,24,23,22,26,25, 16]; however they have not formulated a satisfactory and general solution. Most robotics techniques capitalized on computing 2D workspace cross sections for manipulators with specialized geometries. Two dimensional workspace cross sections are not adequate for manipulators with joint limits since the workspace is not symmetric. In addition, they force the user to memorize the data set by looking at multiple displays before making an interpretation or a decision. Finally, two dimensional workspace cross sections cannot be used in planning systems or computer aided design applications.

2 Background Review

The first efforts to compute the manipulator workspace, based on its kinematic geometry, started in the mid 1970's [18,21]. These proved that the extreme distance line between a chosen point on the first joint axis and the center point of the hand/end effector (extreme reach) intersects all intermediate joint axes of rotation. However, the above result is not valid if:

1. Any intermediate joint axis is parallel to the extreme distance line, or when two joint axes intersect.

2. Any joint is not ideal (has limits).

Kumar and Waldron [13] presented another algorithm to compute the manipulator's workspace. In their analysis, an imaginary force is applied to the reference point at the end effector in order to achieve the maximum extension in the direction of the applied force. The manipulator reaches its maximum extension when the force's line of action intersects all joint axes of rotation (since the moment of the force about each axis of rotation must be zero). Every joint of the manipulator can settle in either of two possible positions under the force action. Hence, this algorithm results in 2^{n-1} different sets of joint variables for a manipulator of n joints in the direction of the applied force. Each set of joint variables results in a point on the workspace boundary. The concept of stable and unstable equilibrium is used to select the set of joints variables that results in the maximum extension in the force direction. The above algorithm is used to generate a shell of points which lie on the workspace boundary by varying the direction of the applied force on unit sphere. This algorithm has exponential time complexity and only deals with manipulators that have ideal revolute joints.
Tsai and Soni [24] developed another algorithm to plot the contour of the workspace on an arbitrarily specified plane for a manipulator with n revolute joints. The robot hand is moved to the specified plane, then the tip of the hand is moved on the plane until it hits the workspace boundary, and finally the workspace boundary is traced by moving the hand from one position to its neighbor. Each of these three subproblems is formulated as a linear programming problem with some constraints and bounded variables (to account for the joint limits). Accordingly, this algorithm suffers from the limitations of not only being restricted to computing 2D workspace cross section but also high computational cost.

[1] An agent can be modeled as a redundant articulated chain.

Yang [26] and Lee [14] presented algorithms to detect the existence of holes and voids in the manipulator's workspace. A workspace is said to have a hole if there exist at least one straight line which is surrounded by the workspace yet without making contact with it. The hole in a donut is a simple example for the above definition. A workspace is said to have a void if there exist a closed region R, buried within the reachable workspace, such that all points inside the bounding surface of R are not reached by the manipulator. Gupta [9,10] classified voids into two different types. The first one, called *central*, occurs around the first axis of rotation and is like the core of an apple. The second type, called *toroidal* or *noncentral*, occur within the reachable workspace and are like a hollow ring. He [9] also presented qualitative reasoning about the transformation of holes to voids and vice versa. Both the qualitative method developed by Gupta [9] and the analytical one developed by Yang and Lee [26,14] are based on mapping the workspace from the distal link to the proximal one and studying the relationship between the generated workspace $W_k(P)$ and the new axis of rotation Z_{k-1}. $W_k(P)$ is defined as the workspace generated by the point P while sweeping joints $k+1, ..., n$ through their entire range while holding axis k fixed. The necessary condition for $W_k(P)$ to contain a void is that $W_{k+1}(P)$ has a hole. However, this is by no means a sufficient condition since $W_k(P)$ may contain no void while $W_{k+1}(P)$ may have a hole depending on the relative position of the axis Z_k and $W_{k+1}(P)$. Yang [26] and Lee [14] classified the geometrical relationships between Z_k and $W_{k+1}(P)$ into three distinct cases and developed an algorithm for detecting voids in the workspace. They [26,14] also studied the conditions for the existence of holes and voids for manipulators with limited joints. If a manipulator has a limited joint $(k+1)$, then the workspace $W_{k+1}(P)$ will have no hole. Accordingly, the workspace $W_k(P)$ will have no void. In general, the manipulator workspace $W_2(P)$ has a hole as well as a void. However, it is always desirable to have the workspace $W_1(P)$ without any void. This can be solely achieved by proper relative positioning of the first and the second axes of rotation. Accordingly, this algorithm can be better used to compute the manipulator's parameters that result in a workspace without holes or voids. The iterative nature of the above algorithm makes it impossible to implement it in parallel since the computation of $W_k(P)$ can not proceed until $W_{k+1}(P)$ is computed. Lee [14] also introduced a kinematic performance index, called VI (volume index of manipulator workspace). It is defined to be the ratio of a manipulator's workspace volume to the cube of the manipulator's total length. This performance index indicates the effectiveness of link length on the creation of reachable workspace. They used that index to compare different commercial robots.

Tsai [22] presented another algorithm, based on the theory of reciprocal screws. In contrast to the above algorithms that only compute workspace points, this algorithm traces 2D workspace boundary for a given manipulator. The use of the reciprocal screws theory has made computing piecewise continuous boundary that consists of straight line segments and circular arcs possible. The manipulator's workspace is computed by performing the union operation on all the workspaces of the manipulator's aspects. An aspect of a robot is interpreted as a set of joint variables such that the manipulator can reach points inside the workspace at one configuration without hitting a joint limit [22]. The computed workspace has interior surfaces which are the boundaries of aspects. This algorithm is limited to manipulators which do not have holes or voids in their workspaces. In addition to this serious limitation, we believe that this algorithm is very costly to use in solving the *workspace synthesis problem* and can be better utilized in solving the *workspace analysis problem.* Both of these problems and the algorithms for solving them are discussed in detail in the next section. Tsai [22] also provided an algorithm for computing the workspace geometric properties such as its volume, the

location of the centroid, and its moments of inertia about the three principal axes. These properties can be used to characterize the workspace shape as is commonly done in robot vision systems. Korien [11] did pioneering work in creating 3-D reach volumes by taking polyhedral unions of reach polyhedra, working along an articulated chain from distal joint inwards. The major drawbacks of his approach are:

1. *High computational cost:* The polyhedral unions are very difficult to perform once they become many-sided.

2. *Error-prone*: the polyhedral union becomes sensitive to numerical imprecision and geometric ambiguities (non-planar facets, gaps) as the unions chopped the polyhedra into smaller and smaller fragments.

3. *Conservative approximation*: The above problems cause too few positions of the swept polyhedra to be taken for the unions.

4. *Impossible parallel implementation:* This is because of the fact that the computation at each link depends on the computation at the previous one.

2.1 Transformation Arithmetic

In this section, we describe the Denavit-Hatenberg [17,5] notations that are commonly used to describe articulated chains such as a robot manipulator or a human limb. An articulated chain is a series of links connected with either revolute or prismatic joints[2]. We assume without loss of generality that each joint has one degree of freedom. A joint with m degrees of freedom can be modeled as m joints connected with links of zero length. The first link of the articulated chain (link 1) is connected to the base (link 0) by joint 1[3]. The final link, denoted by the end effector (link n), has no joint at its end. Each link (i) has two dimensions: the *length* (a_i) and the *twist* (α_i). The length (a_i) is defined to be the common normal distance between the two axes of j_i and of j_{i+1}. The twist (α_i) is defined to be the angle between the two axes of j_i and j_{i+1} in a plane perpendicular to a_i. The joint axis i has two normals to it, one for link $i-1$ and one for link i. The distance between the normals along the joint i axis is denoted by d_i and the angle between the normals measured in a plane normal to the axis is denoted by θ_i. In the case of a revolute joint θ_i is called the *joint variable* and the other three fixed quantities (d_i, a_i, α_i) are called the *link parameters*. On the other hand, d_i is the joint variable for a prismatic joint and the other three fixed quantities $(\theta_i, a_i, \alpha_i)$ are called the link parameters. Hence, any articulated chain can be described kinematically by giving the values of the four quantities for each link[4]. The human body contains only revolute joints; however, for each revolute joint, the joint variable θ_i typically has a lower limit θ_{il} and an upper limit θ_{iu}. In order to be able to describe the relationship between two consecutive links, a coordinate frame is attached to each link as illustrated in figure 1. A homogeneous transformation matrix (A) is used in order to describe the relationship between consecutive frames. We are going to compute the elements of the matrix (A) for both revolute and prismatic joints since these elements depend on the joint type.

[2]The human body contains only revolute joints.
[3]The base is not considered one of the articulated chain n links.
[4]Two of these values describe the link itself while the other two describe its connection to a neighboring link.

2.1.1 Revolute Joints

In this case the joint variable is θ_i. The origin of the frame attached to link i is set to be at the intersection of the common normal between the axes of j_i and j_{i+1} and the axis of joint $i+1$. If the joint axes intersect, we set the frame's origin to be at the point of intersection. In case of parallel joint axes, we choose the origin such that the joint distance, for the next link whose coordinate origin is defined, is equal to zero. The z axis for link i, denoted by Z_i, is coincident with the axis of joint $i+1$. The x axis for link i, called X_i, is aligned with the common normal a_i in the direction from joint i to joint $i+1$. In the special case of intersecting joint axes, X_i is chosen normal to the plane of Z_{i-1} and Z_i. In the case of revolute joint, θ_i, is zero when x_{i-1} and x_i are parallel and in the same direction. Figure 1 illustrates the above parameters.
The relationship between successive frames i-1, i can be established by the following translation and rotation homogeneous transformations:

1. Rotation about Z_{i-1} by an angle θ_i, Rot (Z_{i-1}, θ_i).
2. Translation along Z_{i-1} by a distance d_i, Trans(0,0,d_i).
3. Translation along the rotated $X_{i-1} = X_i$ by a distance a_i, Trans(a_i,0,0).
4. Rotation about X_i by an angle α_i, Rot (X_i, α_i).

Accordingly, the coordinate transformation equation can be expressed as:
$\{Y_i\} = A_i\{Y_{i+1}\}$
where
$\{Y_i\} = (X_i, Y_i, Z_i, 1)^T$.
$\{Y_{i+1}\} = (X_{i+1}, Y_{i+1}, Z_{i+1}, 1)^T$.
A_i= Rot (Z_{i-1}, θ_i) Trans(0,0,d_i) Trans(a_i,0,0) Rot (X_i, α_i).

Accordingly, the transformation matrix is:

$$A_i = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.1.2 Prismatic Joints

In this case the joint variable is d_i. The joint moves in the direction of its axis. Accordingly, the length a_i has no meaning and is set to zero. The origin of the coordinate frame attached to the prismatic joint is chosen to be coincident with the next defined link origin. As in revolute joints, the z axis for link i, denoted by Z_i, is coincident with the axis of joint $i+1$. The x axis for link i, called X_i, is chosen to be normal to the plane of Z_{i-1} and Z_i. The zero position is defined to be the one when d_i is zero. Accordingly, the A matrix for prismatic joint is:

$$A_i = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & 0 \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We have discussed the relationship between two consecutive links and illustrated how to compute the matrix (A) elements for both revolute and prismatic joints. Hence, the n transformation matrices , $A_1A_2...A_i...A_{n-1}A_n$, can be computed for any articulated chain $(l_1, l_2, ..., l_n)$ with either revolute or prismatic joints. Each of the A_i's is a function of the joint variable θ_i or d_i according to the joint type. The description of the coordinate frame attached to the end effector (link n) with respect to the base, denoted by T_n, is given by:
$T_n = A_1A_2...A_i...A_{n-1}A_n$.
The reachable workspace boundary for an articulated chain that has three links or less can be represented by explicit equations [23]. Unfortunately, the reachable workspace boundary for an articulated chain that has more than three degrees of freedom is very hard to describe by explicit equations [24]. The computational complexity of the workspace problem is at least NP-hard [3,1,4, 2]. We decompose the 3D workspace problem into two phases: workspace point computation and visualization. Each phase can be implemented in parallel.

3 Workspace Point Computation

The workspace point computation phase is implemented by using forward kinematics. The forward kinematics problem, which can be solved rather trivially, is the static geometrical problem of computing the position and orientation of the end effector of the articulated body. Specifically, the forward kinematics problem is to compute the position and orientation of a distal segment of the body given the joint positions of the segments proximal to it.
In this section, we present algorithms that are based on forward kinematics. A dense set of reachable points is computed by positioning each degree of freedom in every possible position. These points don't necessarily lie on the workspace envelope (boundary.) If the application only requires computing the reachable workspace envelope, an edge detection algorithm is used to obtain the workspace boundary. The following procedure summarizes this approach:
Procedure 3.1 Point Computation Based on Direct Kinematics.

1. Get the list of active degrees of freedom DOF's that are associated with those joints in the path from the end effector (distal linkage) to the proximal one.

2. Compute a joint angle for each DOF in the above list such that it lies within the given joint limits. This step can be done either by placing the DOF randomly between its associated limits or by uniformly stepping through its limits in each iteration.

3. Compute the end effector position that corresponds to the configuration from step two.

4. Repeat step two and three to compute the required number of points.

5. If the application is surface computation, call an edge detection procedure to compute the set of the useful points (that lie on the boundary).

The fifth step can be implemented by different algorithms. There are two different classes of algorithms that can be used to compute the boundary based on the surface fitting module.

1. If the surface fitting module is going to compute the surface from a set of 2-D contours, the set of boundary points can be computed as follows:

(a) Compute the reachable cube dimensions which is the minimum cube that surrounds all the computed points. This is done by computing X_{min}, X_{max}, Y_{min}, Y_{max}, Z_{min}, Z_{max} for the set of the computed points. This step requires $O(n)$ time where n is the number of the generated points.

(b) Compute the number of contours (noc) as follows:

$$noc = ((Z_{max} - Z_{min})/res_z) + 1 \tag{1}$$

(c) Compute the boundary for each contour as follows:

i. Allocate an array (A) of size M, N where:

$$M = ((X_{max} - X_{min})/res_x) + 1. \tag{2}$$
$$N = ((Y_{max} - Y_{min})/res_y) + 1. \tag{3}$$

ii. Mark each cell (or grid element) in the array A with 1 if it contains a computed workspace point and mark it with 0 if doesn't contain one.

iii. Compute the set of boundary points by testing the corresponding cells in the array A. An element of array A lies on the boundary if it has a different value from its neighbor, i.e.,
$A(i,j) \neq A(i,j+1)$, $A(i,j) \neq A(i,j-1)$, $A(i,j) \neq A(i+1,j)$, or$A(i,j) \neq A(i-1,j)$.
(This assumes that the input points are sufficiently dense).

2. If the surface fitting module is going to compute the 3D reachable surface directly (i.e without computing 2D contours), then the first and the second steps of the above algorithm are applied and the test for the boundary condition in step three is extended.

The above procedure can be implemented in parallel by assigning different processors to compute workspace points at step 3 in procedure 3.1.

4 Workspace Visualization

This module constructs a surface that encompasses the workspace points that were computed by the workspace point generation module. We have developed an algorithm that accepts the workspace contours computed by the direct kinematics algorithm. The algorithm can be summerized in four steps:

Step 1 Region Filling. This step involves determining the number of regions in a given workspace contour. The number of holes and voids in the given workspace contour can be determined. Region filling algorithms are a common graphics utility and are widely used in paint programs [20,15,19,7,6]. A region is a collection of pixels. There are two types of regions: 4-connected and 8-connected. A region is 4-connected if every 2 pixels can be joined by a sequence of pixels using only up, down, left, or right moves. A region is 8-connected if every 2 pixels can be joined by a sequence of pixels using up, down, left, right, up-and-left, up-and-right, down-and-left, or down-and-left moves. Region filling algorithms start with a given seed point (x, y) and set this pixel and all of its neighbors with the same pixel value to a new pixel value. A good

region-filling algorithm is one that reads as few pixels as possible. We use Fishkin's algorithm to compute the number of regions in a given workspace cross-section (contour). We search the contour for a reachable workspace cell (marked with 1) and use it as a seed point. The set of all cells connected to the seed point comprise a reachable region. The region filling algorithm sets those cells to a new value that greater than 2. The region filling algorithm is called as many times as necessary in order to set different regions with unique values.

Step 2 Boundary Detection. The purpose of this step is to compute the boundary of different regions in a workspace cross-section. This can be achieved by testing the neighbors for each cell in the workspace cross-section. An array element $A(i,j)$ of the workspace cross-section is considered a boundary cell if it has a different value from its neighbor, i.e.,
$A(i,j) \neq A(i,j+1)$, $A(i,j) \neq A(i,j-1)$, $A(i,j) \neq A(i+1,j)$, or$A(i,j) \neq A(i-1,j)$.

Step 3 Contour Tracing. This step computes the edges that connect the boundary points for a given region.

Step 4 Triangulation. This step constructs the 3D workspace by tiling adjacent contours with triangles. We have used the Fuchs' algorithm [8] that interpolates the triangular faces between parallel slices in order to construct the 3D workspace surface from the different cross sections. Each of the above steps can be implemented in parallel by assigning different processors to compute different contour sections.

5 Conclusions

We have presented a system for computing 3D workspaces for articulated chains not only those with redundant degrees of freedom but also those with joint limits. The first phase of that system computes workspace points in parallel. The second phase fits a 3D surface around the volume that encompasses the workspace points computed by the first phase.

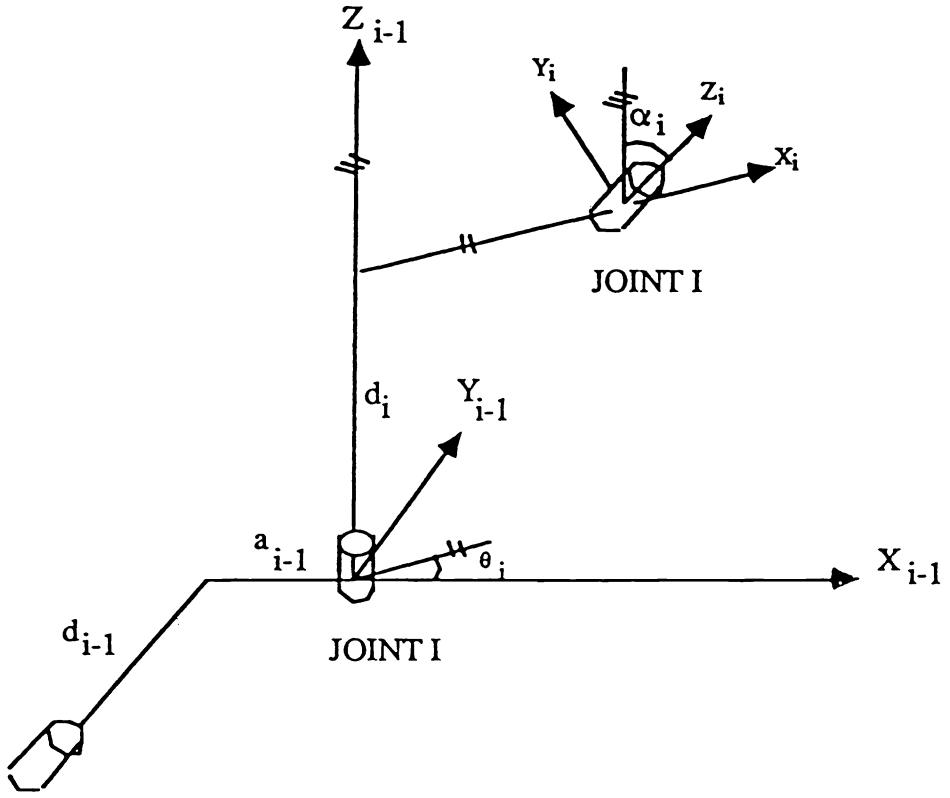

Figure 1: The relationship between consecutive links.

References

[1] T. Alameldin. Three Dimensional Workspace Visualization for Redundant Articulated Chains. PhD thesis, University of Pennsylvania, 1991.

[2] T. Alameldin, N. Badler, and T. Sobh. *An Adaptive and Efficient System for Computing the 3-D Reachable Workspace.* In *Proceedings of IEEE International Conference on Systems Engineering*, 1990.

[3] T. Alameldin, M. Palis, S. Rajasekaran, and N. Badler. *On the Complexity of Computing Reachable Workspaces for Redundant Manipulators.* In *Proceedings of SPIE Intelligent Robots and Computer Vision IX: Algorithms and Techniques*, 1990.

[4] T. Alameldin and T. Sobh. *On the Evaluation of Reachable Workspace for Redundant Manipulators.* In *Proceedings of the Third International Conference of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, 1990.

[5] J. Craig. Introduction to Robotics Mechanics & Control. Addison Wesley, 1986.

[6] K. Fishkin. *Filling A Region In a Frame Buffer.* In A. Glassner, editor, *Graphics Gems*, pages 278–284, Academic Press, Cambridge, Mass., 1990.

[7] K. Fishkin and B. Barsky. *An Analysis and Algorithm for Filling Propagation.* In *Proceedings Graphics Interface*, pages 203–212, 1985.

[8] H. Fuchs, Z. Kedem, and S. Uselton. *Optimal Surface Reconstruction from Planar Contours. Communications of The ACM*, 693–702, October 1977.

[9] K. Gupta. *On the nature of Robot Workspace. International Journal of Robotics Research*, 5:112–122, 1986.

[10] K. Gupta and B. Roth. *Design Considerations for Manipulator Workspace. ASME Journal of Mechanical Design*, 104:704–711, October 1982.

[11] J. Korein. A Geometric Investigation of Reach. MIT Press, Cambridge, MA, 1985.

[12] A. Kumar. Characterization of Manipulator Geometry. PhD thesis, University of Houston, 1980.

[13] A. Kumar and K. Waldron. *The Workspace of a Mechanical Manipulator. ASME Journal of Mechanical Design*, 103:665–672, July 1981.

[14] T. Lee and D. Yang. *On the of Manipulator Workspace. Journal of Mechanisms, Transmissions, and Automation in Design*, 105:70–77, March 1983.

[15] M. Levoy. *Area Flooding Algorithms.* In *SIGGRAPH '82 2D Animation Tutorial*, 1982.

[16] C. Mujabbir. Workspaces of Serial Manipulators. Master's thesis, University of Pennsylvania, 1987.

[17] R. Paul. Robot Manipulators: Mathematics, Programming, and Control. MIT Press, Cambridge, Mass., 1981.

[18] B. Roth. *Performance Evaluation of manipulators from a Kinematics Viewpoint. NBS Special Publication*, 39–61, 1975.

[19] U. Shani. *Filling Regions in Binary Raster Images: A Graph-Theoretic Approach.* In *SIGGRAPH '80 Conference Proceedings*, pages 321–327, 1980.

[20] A. Smith. *Fill Tutorial Notes.* In *SIGGRAPH '82 2D Animation Tutorial*, 1982.

[21] D. Sugimoto. *Determination of Extreme Distances of a Robot Hand. ASME Journal of Mechanical Design*, 103:631–636, July 1981.

[22] M. Tsai. Workspace Geometric Characterization and Manipulability of Industrial Robots. PhD thesis, Ohio State University, 1986.

[23] Y. Tsai and A. Soni. *Accessible Region and Synthesis of Robot Arms. ASME Journal of Mechanical Design*, 103:803–811, October 1981.

[24] Y. Tsai and A. Soni. *An Algorithm For the Workspace of a General n-R Robot. ASME Journal of Mechanical Design*, 105:52–57, July 1983.

[25] R. Vijaykumar. Robot Manipulators - Workspaces and Geometrical Dexterity. Master's thesis, Ohio State University, 1985.

[26] D. Yang and T. Lee. *On The Workspace of Mechanical Manipulators. Journal of Mechanisms, Transmissions, and Automation in Design*, 105:62–69, March 1983.