Rule-based Generation of Motion Control Software for General Serial-Link Robot Manipulators

M. Wenz
Institute for Process Control and Robotics (IPR)
Universität Karlsruhe (TH)
Karlsruhe, Germany
e-mail: mwenz@ira.uka.de,

H. Wörn
Institute for Process Control and Robotics (IPR)
Universität Karlsruhe (TH)
Karlsruhe, Germany
e-mail: woern@ira.uka.de

Abstract[1]

Monolithically structured robot controls can only be adapted and enhanced with high efforts. Therefore we develop a configuration system and a graphical user interface in order to configure the robot control on the fly. Within the envisaged scenario the operator is able to configure the motion control according to the mechanical structure of a robot by just pushing of "a button". For the flexible configuration of robot control systems knowledge-based approaches are pursued. The motion control of a simulated robot is produced by the simple combination of components. In particular we automatically configure the forward and backward kinematics on the basis of a declarative description of a robot, which among other things indicates the number of joints. The main goal of our project is the development of a generalized software architecture that applies to all classes of robots.

1. Introduction

Motion control is one of the main research areas in robotics. The development of motion control software for serial robots has traditionally been a longsome process that was generally a custom approach for each robot type. A central problem is the determination of the inverse kinematics of serial manipulators with arbitrary geometry. It has been shown that the joints of such a general 6-degree-of-freedom manipulator can orient themselves in up to 16 different configurations for a given position and orientation of the tool centre point [1, 2]. However numerical solutions like the Newton-Raphson technique to the inverse kinematics which can be used for arbitrary geometries cannot find all these solutions. In contrary geometric and algebraic methods can find all these solutions, but they cannot be applied to all robot geometries, especially they cannot be applied to hyper redundant manipulators with an infinite number of solutions. As a result, most industrial manipulators are designed so that a closed form solution exists. To simplify the development of motion control software most industrial robots have only revolute or prismatic joints and orthogonal, parallel and/or intersecting joint axes (instead of skew arranged joint axes) and often three consecutive axes intersect in a common point. However the development of motion control software is still an individual programming effort for a particular robot employing computational tools, e.g. MatLAB™ with little or no integration possibilities.

In the next section we review related work and the current state of the art in detail. Section 3 describes our research project about automatic configuration of robot control software. An example of how to automatically generate the motion control of a 4-degree-of-freedom robot will be given at the end of the paper.

2. Related Work

A proprietary and not universal control system is difficult to modify and makes it hard to integrate new hard- and software components. Therefore robotics research today focuses on developing systems that exhibit modularity, flexibility and intelligence. A major issue in the field is software reuse. Robot software is inherently coupled to the mechanical structure and to the underlying hardware making it difficult to adapt and to reuse the software. As a consequence there were a series of initiatives started in order to produce a functional basis for robot control systems.

OROCOS (Open Robot Control Software / Open Realtime Control Services) [3] is a European open source software project coordinated by Universiteit Leuven, Belgium. The project is based on design patterns as well as on object-oriented programming methods. OROCOS has a highly modularized structure, where it is easy to

[1] *Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CSIT copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute for Contemporary Education JMSUICE. To copy otherwise, or to republish, requires a fee and/or special permission from the JMSUICE.*

Proceedings of the 8th International Workshop on Computer Science and Information Technologies CSIT'2006
Karlsruhe, Germany, 2006

connect and to replace modules in order to master the software complexity problem.

OSCAR (Operational Software Components for Advanced Robotics) [4, 5] is a project of the Robotics Research Group at University of Texas, Austin. The OSCAR framework is based on an object-oriented design. Components developed in OSCAR are thus reusable and applicable across different robot types. OSCAR also offers functionalities to deal with redundant and fault tolerant advanced robots. The OSCAR framework has been developed and used for several years, and counts NASA and the Department of Energy (DOE) amongst its users.

ROBOOP (A robotics object oriented package in C++) [6] is a library for robotics synthesis and simulation developed at the Ecole Polytechnique de Montreal. The robot classes include a class to compute the forward kinematic model using a recursion relation and a class to compute the inverse kinematic model using a Newton-Raphson technique. The project is under the terms of the GNU LGPL and a source code distribution can be downloaded. JROBOOP is an open Java package inspired by ROBOOP and developed by the Robotics and Automation Group of the University of Naples.

The Matlab Robotics Toolbox [7, 8] was developed at the CSIRO ICT Centre Australia by P. Corke and provides the kinematics and dynamics models for well known robots such as the Unimate Puma 560 and the Stanford arm. The toolbox cannot derive symbolically the closed-form solution of the inverse kinematics in case that the analytical model of the robot exists. Instead it calculates an iterative numerical solution which can be slow and provides only limited control over the particular solution that will be found.

Robotica [9] is a Mathematica package developed at the University of Illinois at Urbana-Champaign. Its input is a text file containing the Denavit-Hartenberg parameters and dynamics data describing the robot to be analyzed. Robotica generates the forward kinematics, the Jacobian matrix and the complete Lagrangian dynamic model but there are no functions for trajectory planning or inverse kinematics. Results can be generated in a purely symbolic form and thus Robotica requires Mathematica. But because the project is no longer supported or updated it will not be compatible with future Mathematica versions.

3. Organic Robot Control

Within a subproject of the priority programme 1183 "Organic Computing" of the German Research Foundation (DFG) a software architecture for a robot controller is developed with emphasis on self-configuration and self-organization features. In particular a configuration system and a graphical user interface are developed in order to configure the robot control on the fly. In general, a serial robot consists of a number of rigid links connected with joints. The operator specifies the mechanical structure of a particular robot and then let the configurator automatically generates the motion control. This self-configuration is based on domain knowledge and on rules. The configurator opens up numerous selection and combination possibilities:

- Kinematic structure of the first three joints, e.g. SCARA, cartesian, cylindrical, spherical or articulated robot.
- Number of joints ($n = 4 \ldots 6$), each joint can be rotatory or prismatic. Four and five degrees-of-freedom robots are usually equipped with symmetric tool pieces.
- Two joints can be parallel or at right angle in the zero position of the robot. When two joints are arranged at right angle then they can intersect or cannot intersect. When two joints are arranged parallel then there can be some space between them or they can coincide.
- Geometric dimensions, arm lengths and workspace.
- Interpolation clock, velocity profile and interpolation algorithms that should be supported, e.g. point-to-point, linear, circular and spline.

Besides the automatic generation of motion control software the user can integrate his own software components over a specified API.

3.1. Denavit-Hartenberg Convention

We require a systematic manner for modeling the robot's geometry. We use the Denavit-Hartenberg (DH) convention proposed in 1955 [10], because it is the most compact general representation of the robot's geometry. A minimum number of parameters describe the link geometry. At first the joint axes are identified and each joint is assigned a coordinate frame according to the following rules.

- If the joint axis $i+1$ is revolute, the z_i axis is directed along the axis of rotation as followed by the right hand rule.
- If the joint axis $i+1$ is prismatic, the z_i axis is directed along the linear motion.

We define frame 0 as the base frame and begin an iterative process in which we define frame i using frame $i-1$, beginning with frame 1. In order to determine the x_i axis and the origin of frame i we distinguish four cases:

1. The axes z_{i-1}, z_i are arranged at right angle and intersect: The origin of the ith coordinate frame is located at the intersection of joint z_i and z_{i-1} axis. The x_i axis is directed along the common normal. The positive direction of x_i is arbitrary.
2. The axes z_{i-1}, z_i are arranged at right angle and do not intersect: The origin of the ith coordinate frame is located at the intersection of joint z_i axis and the

common normal between joint axis z_i and z_{i-1}. The x_i axis is directed along the common normal.

3. The axes z_{i-1}, z_i are parallel: The x_i axis points along the common perpendicular from the z_{i-1} to the z_i axis. The origin of frame i can be chosen anywhere on z_i and is set to the point where the x_i axis that passes through the origin of the i-1th frame intersects the z_i axes.

4. The axes z_{i-1}, z_i are coinciding: The x_i axis can be chosen arbitrary and is set to the x_{i-1} axis and the origin of the ith frame is set to the origin of the i-1th frame.

When the axes z_{i-1} and z_i skew it is necessary to assign the frames by hand, but this case is very seldom. The y_i axis is chosen to create a right-handed coordinate system, i.e. the y_i axis is simply the cross product of the z_i and x_i axes.

In order to describe the relative location of frame i with respect to the previous frame i-1 the four DH-parameters (θ_i, d_i, a_i, α_i) are associated with each joint pair.

1. The joint angle θ_i is the angle between the x_{i-1} and x_i axes about the z_{i-1} axis.
2. The link offset d_i is the distance between the x_{i-1} and x_i axes along the z_{i-1} axis and respectively the perpendicular distance between x_{i-1} and x_i axes.
3. The link length a_i is the distance between the z_{i-1} and z_i axes along the x_i axis and respectively the perpendicular distance between z_{i-1} and z_i axes.
4. The link twist α_i is the angle from the z_{i-1} axis to the z_i axis about the x_i axis.

There is no rotation or translation about the y_{i-1} or y_i axis. In order to automatically determine the DH parameters we brought following expert knowledge into rule form:

- If the ith joint is rotatory, then θ_i is a variable, not a parameter.
- If the ith joint is prismatic, then d_i is a variable, not a parameter.
- The parameters a_i and α_i are always fixed parameters.
- If the ith joint is prismatic and the axes x_i and x_{i-1} are parallel then the parameter θ_i is equal 0° (θ_i=180° when the axes are anti-parallel).
- If the ith joint is rotatory and the axes x_i and x_{i-1} intersect then the parameter d_i is equal 0.
- If the axes z_i and z_{i-1} intersect then the parameter a_i is equal 0.
- If the axes z_i and z_{i-1} are parallel then the parameter α_i is equal 0° (α_i =180° when the axes are anti-parallel).

After setting up a coordinate frame for every joint of the robot we can automatically compute the DH-parameters which cannot be concluded with the above rules. Once we do this, we establish a set of matrices that transform one coordinate frame to the next one.

For two frames positioned in space, the first can be moved into coincidence with the second by a sequence of one rotation, two translations and one rotation. Thus, each homogenous transformation matrix from one coordinate frame to the next frame is represented as a product of four basic transformations.

$$T_{i-1,i} = Rot(z_{i-1},\theta_i) * Trans(z_{i-1},d_i) * Trans(x_i,a_i) * Rot(x_i,\alpha_i). \quad (1)$$

3.2. Forward Kinematics

For serial manipulators, the forward kinematics is straightforward to derive. We formulate the forward kinematics using four-by-four homogeneous transformation matrices. The matrix $T_{0,N}$ describing the position and orientation of the end-effector relative to the base is calculated as the product of the N intermediate transformations matrices $T_{i-1,i}$ from joint i-1 to joint i where N is the number of joints of the robot:

$$T_{0,N} = T_{0,1} * T_{1,2} * \ldots * T_{n-1,n} = \prod_{i=1}^{N} T_{i-1,i} \quad (2)$$

In this context, the joint variables are given, and then the position and orientation of the end-effector is automatically computed. There always exists a unique solution.

3.3. Inverse Kinematics

The motion control requires the solution of the inverse kinematics, which computes the joints variables given the position and orientation of the end-effector. This problem is much more difficult than the forward kinematics problem, because a set of nonlinear equations has to be solved. We use the same equations as for the forward kinematics, but now the TCP frame is specified and the joint coordinates are unknown. The equations are tightly coupled with multiple unknowns and with many trigonometric functions. A closed form solution can not always be derived. When dealing with general manipulators the equations can become very large and difficult to solve requiring advanced symbolic transformation techniques.

We are much more interested in finding a closed form solution of the inverse kinematics rather than a numerical solution. There are two major reasons why closed form solutions are preferable. At first, they can be much faster calculated than iterative numerical ones. This is especially important when real-time constraints have to be satisfied. Iterative solutions require a lot of computations and are also more prone to numerical errors which can occur than analytical solutions. At second, robots with many rotatory joints may have several solutions for one given Cartesian position and orientation. These solutions often exhibit various forms of symmetry. Having multiple solutions allows us to choose a particular solution among them.

It is also possible that there exists no solution for the inverse kinematics, because any robot only has a finite reach. When the desired TCP location is out of the workspace the inverse kinematics has no solution. Therefore each obtained solution is checked whether it satisfies all constraints regarding the ranges of possible joint motions. Solutions that violate the joint limits are discarded. Of the remaining valid solutions, usually the one closest to the current manipulator configuration is chosen.

There does not exist always a closed form solution for the inverse kinematics. In his 1968 Ph.D. thesis [11], D. L. Pieper enumerated special cases in which a closed form solution is feasible. These cases include any serial manipulator with six revolute joints when three consecutive joints intersect or when three consecutive joints are parallel. In particular Pieper's solution applies to robots with spherical wrist, where the solution can be found by decoupling the Cartesian position and orientation. Tourassis [12] also identified some special cases of manipulator design where a decoupling of the robot geometry is guaranteed and thus the inverse kinematics can be solved in closed form.

We use the algorithm of Paul [13] for determining a closed kinematics solution. Thereby the equations of the forward kinematics are arranged in many different ways in order to isolate unknown joint variables. Although there are only 12 equations and up to 6 unknown joint variables, they can be rewritten in many different ways. Each side of the direct kinematics equations is pre- and postmultiplied by the inverse of the transformation matrices $T_{i-1,i}^{-1}=T_{i,i-1}$. For a 4-degree-of-freedom manipulator we generate the following equation set.

$$\begin{array}{ccc}
T_{0,1} * T_{1,2} * T_{2,3} * T_{3,4} & = & T_{0,4} \\
T_{0,1} * T_{1,2} * T_{2,3} & = & T_{0,4} * T_{4,3} \\
T_{1,2} * T_{2,3} * T_{3,4} & = & T_{1,0} * T_{0,4} \\
T_{0,1} * T_{1,2} & = & T_{0,4} * T_{4,3} * T_{3,2} \\
T_{1,2} * T_{2,3} & = & T_{1,0} * T_{0,4} * T_{4,3} \\
T_{2,3} * T_{3,4} & = & T_{2,1} * T_{1,0} * T_{0,4} \\
T_{0,1} & = & T_{0,4} * T_{4,3} * T_{3,2} * T_{2,1} \\
T_{1,2} & = & T_{1,0} * T_{0,4} * T_{4,3} * T_{3,2} \\
T_{2,3} & = & T_{2,1} * T_{1,0} * T_{0,4} * T_{4,3} \\
T_{3,4} & = & T_{3,2} * T_{2,1} * T_{1,0} * T_{0,4}
\end{array} \tag{3}$$

These are of course redundant transformations of the same set of equations. The generated number of equations is quite large: 12*n*(n+1)/2, where n is the number of transformation matrices. However, many equations do not contribute to a solution to the inverse kinematics problem. Since the Denavit-Hartenberg-Convention tends to isolate the joint variables we can also get some equations with fewer unknowns. In the case of parallel revolute joints we use the sums of the joint angles as intermediate variables in order to simplify the equations. Thereby we make use of the trigonometric addition theorems. The equations are asserted into the working memory of our rule-based system. These equations are difficult to solve, although there may be only a single variable in an equation. However this variable may appear in several different trigonometric functions like sine and cosine.

To overcome this problem we use a pattern based transformation technique and a knowledge base about mathematical solutions. If an equation is matched with a pattern in the knowledge base then the corresponding transformation is done. In particular the solutions for a series of elementary trigonometric equations are directly generated. For example Wolovich [14] determined the unique solution for the following two equations in one unknown θ given the known terms a, b, c and d

$$\begin{array}{l}
a * \cos\theta - b * \sin\theta = c \\
a * \sin\theta + b * \cos\theta = d \\
\Rightarrow \theta = \arctan_2(a * d - b * c, a * c + b * d)
\end{array} \tag{4}$$

Note that $a^2+b^2=c^2+d^2$. When both arguments of the arctangent function are zero, θ is not defined and the manipulator is in a singular configuration. Paul [15] computed further solutions for various trigonometric equations which we also included in our knowledge base.

3.4. Symbolic Singularity Analysis

A singularity can occur at points where two different inverse kinematic solutions converge, when joint axes become aligned or parallel, or when the boundaries of the workspace are reached. The determination and the avoidance of singularities are another major issue in the motion control of robot manipulators, because at a singularity, the mobility of a manipulator is reduced. The arbitrary motion of the manipulator in a Cartesian direction is usually lost.

The Jacobian matrix is obtained by differentiating the equations of the forward transformation. In singular configurations the Jacobian matrix of a manipulator is singular. Therefore we determine and analyze the Jacobian matrix and in particular the cases when the determinant is zero. All values of joint variables which result in zero or near zero determinants are at singular configurations. However, it is difficult to identify singular configurations for general manipulators. As the manipulator complexity increases, the complexity of the Jacobian matrix determinant also increases.

4. Example Configuration

In this section we configure the motion control of a Scara (Selectively Compliant Assembly Robot Arm) step by step. The Scara is one of the simplest kinematics and is mainly used for "pick-and-place" operations. Since the robot has to push things into other things, the robot must be stiff in the vertical direction.

The RRT-design (Revolute-Revolute-Translational) has two parallel vertical revolute joints; the third one is an anti-parallel prismatic joint for moving the end-effector normal to the plane and the fourth one is a vertical revolute joint for orienting the end-effector. Therefore θ_1, θ_2, d_3 and θ_4 are the variable joint parameters and the other DH-parameters are the constant ones. Since the first and second joint axes are parallel, α_1 is zero. The parameter α_2 is equal 180° since the second and third joint axes are anti-parallel. The third and fourth joint axes are also parallel. In this case α_3 is zero. The parameter a_1 is the length of the perpendicular between the first and second joint axes and a_2 is the length of the perpendicular between the second and third one. Since the third and fourth axes coincide a_3 is zero. The end-effector frame is chosen to be coincident with frame 3 when θ_4 is zero. Therefore d_4, a_4 and α_4 are zero. The complete DH-parameters based on the robot geometry are indicated in following table.

Table 1. DH-parameters of RRT-Scara

Joint i	θi	di	ai	αi
1	θ1	d1	a1	0°
2	θ2	0	a2	180°
3	0°	d3	0	0°
4	θ4	0	0	0°

At next the transformation matrices are symbolically derived from the DH-table. The forward kinematics equations are computed by multiplying the transformation matrices together.

$$\begin{aligned}
n_x &= \cos(\theta_1+\theta_2-\theta_4)\\
n_y &= \sin(\theta_1+\theta_2-\theta_4)\\
n_z &= 0\\
o_x &= \sin(\theta_1+\theta_2-\theta_4)\\
o_y &= -\cos(\theta_1+\theta_2-\theta_4)\\
o_z &= 0\\
a_x &= 0\\
a_y &= 0\\
a_z &= -1\\
p_x &= a_1 * \cos(\theta_1) + a_2 * \cos(\theta_1+\theta_2)\\
p_y &= a_1 * \sin(\theta_1) + a_2 * \sin(\theta_1+\theta_2)\\
p_z &= d_1 - d_3
\end{aligned} \tag{5}$$

In these equations we represent the orientation of the TCP as an orthonormal rotation matrix containing 9 elements. In general, the position and orientation of the TCP frame is given as

$$T_{0,N} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

The complexity of solving the inverse kinematics increases with the number of nonzero DH-parameters. However, the equations in this example are simple enough to solve directly in closed form. Especially the position and orientation variables are decoupled and only five of these 12 equations contribute to the solution. First, we look for solvable equations like those having a single unknown variable. In the last equation p_z is only dependent on the known variable d_1 and the unknown variable d_3, so we can directly solve for d_3. Equations for p_x and p_y are used to determine θ_2 and θ_1. The solution is extracted by pattern matching with our knowledge base. The same is done with the equations for n_x and n_y in order to determine $\theta_1 + \theta_2 - \theta_4$. Hence, we get a solution for θ_4 depending on θ_1 and θ_2.

$$\begin{aligned}
\theta_1 &= \begin{aligned}[t]&\arctan_2(\\ &p_y * (a_2 * \cos\theta_2 + a_1) - p_x * (a_2 * \sin\theta_2),\\ &p_x * (a_2 * \cos\theta_2 + a_1) - p_y * (a_2 * \sin\theta_2))\end{aligned}\\
\theta_2 &= \arctan_2\left(\pm\sqrt{1-\left(\frac{p_x{}^2 + p_y{}^2 - a_2{}^2 - a_1{}^2}{2a_1a_2}\right)^2}\,,\ \frac{p_x{}^2 + p_y{}^2 - a_2{}^2 - a_1{}^2}{2a_1a_2} \right)\\
d_3 &= d_1 - p_z\\
\theta_4 &= \theta_1 + \theta_2 - \arctan_2(n_y, n_x)
\end{aligned} \tag{7}$$

According to these equations there are up to two solutions of the inverse kinematics for a desired position and orientation of the end-effector when ignoring joint limits and the presence of obstacles. These two solutions are referred as elbow-up and elbow-down configurations.

By creating the Jacobian matrix and setting its determinant equal zero, we determine singularities in the robot workspace

$$\begin{aligned}
\det(J) &= a_1 * a_2 * \sin(\theta_2)\\
\det(J) = 0 &\Leftrightarrow \theta_2 = 0° + k * 180°
\end{aligned} \tag{8}$$

Therefore, singular configurations occur when the arm is fully extended or fully contracted. Then it is not possible to move radially either towards or away from the origin in Cartesian space. We have a restricted motion in the plane.

in Cartesian space. We have a restricted motion in the plane.

5. Conclusion and Future Work

In this paper we have presented the structure of a configurator for general serial manipulators. The configurator was developed with the goals of flexibility and efficiency in mind in order to reduce the time to generate customized motion control software. In particular the configurator eliminates the effort of producing the cumbersome kinematic equations manually for each robot type. In the near future we will also implement numerical methods to find a solution to the inverse kinematics problem if a closed form solution does not exist. Furthermore we will also enhance the motion control system with functionalities for trajectory generation in Cartesian and in joint space.

Acknowledgments

The work presented in this paper is supported by a grant from the German Research Foundation (DFG) within the scope of the priority programme ''Organic Computing'', SPP 1183.

References

1. Craig J.J. "Introduction to robotics: mechanics and control". 3rd edition, Addison-Wesley, 2004.
2. Spong M., Hutchinson S., Vidyasagar M. "Robot Modeling and Control". John Wiley and Sons Inc., 2005.
3. Bruyninckx H. "Open robot control software: the OROCOS project". In: *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*. Seoul, Korea, 2001, pp. 2523-2528.
4. Pryor M., Taylor R., Kapoor C., Tesar C. "Generalized software components for reconfiguring hyper-redundant manipulators". *IEEE/ASME Transactions on Mechatronics*, 2002; 7(4): 475-478.
5. March P., Taylor R., Kappor C., Tesar D. "Decision making for remote robotic operations". In: *Proc. of IEEE Conf. on Robotics and Automation*, Vol. 3, 2004, pp. 2764-2769.
6. Gourdeau R. "Object oriented programming for robotic manipulators simulation". *IEEE Robotics and Automation Magazine*, 1997; 4(3): 21-29.
7. Corke P.I. "A Robotics Toolbox for MATLAB". *IEEE Robotics and Automation Magazine*, 1996; 3(1): 24-32.
8. Corke P.I. "A computer tool for simulation and analysis: the Robotics Toolbox for MATLAB". In: *Proc. of the 1995 National Conference of the Australian Robot Association*. Melbourne, Australia, 1995, pp. 319-330.
9. Nethery J.F., Spong M.W. "Robotica: A mathematical package for robot analysis". *IEEE Robotics and Automation Magazine*, 1993.
10. Denavit J., Hartenberg R.S. "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices". *ASME Journal of Applied Mechanics*, 1955; 22: 215-221.
11. Pieper D.L. *The kinematics of manipulators under computer control.* PhD thesis, Department of Mechanical Engineering, Stanford University, 1968.
12. Tourassis V., Ang M. "Task decoupling in robot manipulators". *Journal of Intelligent and Robotic Systems*, 1995; 14(3): 283-302.
13. Paul R., Renaud M., Stevenson C. "A Systematic Approach for Obtaining the Kinematics of Recursive Manipulators Based on Homogeneous Transformations". In: M. Brady and R. Paul (eds) *Robotics Research - The first International Symposium*. The MIT Press, Cambridge, Massachusetts, 1984, pp. 707-726.
14. Wolovich W. "Robotics, Basic Analysis and Design". CBS College Publishing, 1987.
15. Paul R. "Robot Manipulators: Mathematics, Programming, and Control". The MIT Press, Cambridge, Massachusetts, 1981.