AN ABSTRACT OF THE THESIS OF

Marcia M. Harris-Coppola for the degree of Master of Science in Computer Science presented on April 12, 1988.

Title: Analytical and Visual Tools for Robot Kinematics

Abstract approved: Redacted for Privacy

Computer control of a robot arm's motion requires kinematic algorithms for relating the state of a particular arm's joints to the position and orientation of its tool in three-dimensional space. To design such algorithms requires mathematical formulation of the kinematics of the arm. The resulting long, tedious algebraic manipulations suggest a need for computer-aided kinematic analysis, integrated with more conventional robotic tools. In this paper, we address this problem in four steps. The first step is to design a simple, yet sufficiently general, representation of robot arm links, which we call the orthogonal representation. The second step is to design and implement a module to generate the Forward Kinematic Equation automatically in algebraic form for arbitrary robot arm configurations. The third step is to complement the kinematics module with a robot simulator and a graphic display. The fourth step is to attack the generally intractable Inverse Kinematic problem by analyzing frequently-occurring subconfigurations, and then implementing subsolutions from which the entire arm's solution is built.

Analytical and Visual Tools for Robot Kinematics

by

Marcia M. Harris-Coppola

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Completed April 12, 1988

Commencement June, 1988

APPROVED:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented _____

Typed by Marcia Harris-Coppola for Marcia Harris-Coppola

Acknowledgements

I would like to express my sincere thanks to Professor Fred M. Tonge, my thesis advisor, for his encouragement and careful reading of this thesis, and to Dr. Guy W. Cherry, who suggested the idea of applying computer algebra to robot kinematics, for his continual encouragement and assistance. I thank Professors Earl F. Ecklund, Jr., Bella Bose, and David S. Birkes for reading this thesis and suggesting improvements.

I want to express thanks to Dr. Balaji Krishnamurthy for making the facilities of the Computer Research Laboratory of Tektronix, Inc., available to me for this project, and to Dr. Kamal Abdali, Neil Soiffer, Dr. Philip Todd, Eirik Fuller, David Hatcher and Beverly Milhouse for their encouragement and support.

Special thanks are due to Hannah Todd for her constructive criticism, sympathy, practical assistance, and friendship.

Finally, I want to express my appreciation to my husband, Alan, and my daughters, Kate and Elizabeth, for their support and understanding during the period of domestic disruption occasioned by the writing of this thesis.

Table of Contents

List of Figures

Analytical and Visual Tools for Robot Kinematics

Chapter 1

Introduction

1.1 Overview

Computer control of a robot arm's motion requires kinematic algorithms for relating the state of a particular arm's joints to the position and orientation of its tool in three-dimensional space. To design such algorithms (and more generally to analyze the behavior of the arm in space) requires mathematical formulation of the kinematics of the arm. The resulting long, tedious algebraic manipulations of complicated vector equations suggest a need for computer-aided kinematic analysis, integrated with more conventional robotic tools.

In this paper, we address this problem in four steps. The first step is to design a simple, yet sufficiently general, representation of robot arm links, which we call the orthogonal representation. The second step is to design and implement, using orthogonal representation, a module to generate the Forward Kinematic Equation automatically in algebraic form for arbitrary robot arm configurations. The third step is to complement the kinematics module with a robot simulator and a graphic display. The fourth step is to attack the generally intractable

Inverse Kinematic problem by analyzing frequently-occurring subconfigurations, and then implementing subsolutions from which the entire arm's solution is built.

The first step is presented in Chapter 2, where we discuss the representational problems of robot arm kinematics. A notable feature is our descriptive formalism. At several points we have turned away from the notations promulgated by Richard P. Paul's influential work (such as Denavit-Hartenberg homogeneous transformations and generalized link nomenclature), in favor of simpler and more intuitive, and yet sufficiently general, formulations.

The three remaining steps are presented in Chapter 4, where we discuss the implementation of the modules which constitute a new integrated software tool called RobotScribe. The purpose of RobotScribe is to provide algebraic manipulation of kinematic equations, simulation of command-driven robot motion, and graphical three-dimensional display of the robot arm, for robotic arms of nearly any configuration in an integrated, visually-oriented, interactive computing environment. RobotScribe accepts descriptions of arbitrary robot arms and automatically generates their kinematic equations in symbolic form, in an algebraic manipulation environment, and also provides a toolkit for the interactive solution of the Inverse Kinematic problem.

RobotScribe is implemented as an application to problems of robot kinematics of the windows-based computer algebra system MathScribe. The code for RobotScribe is written in the symbolic calculation language REDUCE, with

some enhancements in order to take advantage of the plotting capabilities of MathScribe.

1.2 The Impetus: Aiding Insight in Kinematic Design and Application

The problems of robot design and application fall into three broad areas: planning, control, and kinematics.

Planning is the aspect that is concerned with the robot as an agent of some intelligence. Many robot planning problems are common to high-level organization in a variety of computerized systems. Planning addresses tasks to be performed within environments. It involves programming languages, internal representations of the robot, geometry, and artificial intelligence.

Control is the aspect that is concerned with the robot as a system of actuators. Many robot control problems are common to low-level electromechanical interfacing in a variety of computerized systems. Control involves the electrical and electronic concerns of using feedback to control the actuator at each joint to achieve prescribed motion of the joint. Mechanical concerns of stiffness, natural frequency, and gear backlash relate to control.

Kinematics is the aspect that is concerned with the robot as a mechanical structure in motion. Robot kinematics represents the spatial complexity that is unique to robotic systems. Kinematics is built upon physics, applied mathematics and the mechanical engineering of linkages and dimemsional tolerances. It addresses the position in three-dimensional space of points upon

the manipulator arm, and their velocity and acceleration, as related to the angular (or linear) displacement of the manipulator's joints. Its approach of mathematizing mechanics generalizes to statics and dynamics.

Planning, the high-level aspect, deals with objects and actions in three-dimensional space. Control, the low-level aspect, deals with robot joint behavior. To connect the joint angles and displacements that we can control with the objects and actions that constitute the robot's task, we need the mathematical transformations that are the subject of kinematics.

Compared to most computerized systems, visualizing states of a robot arm can be unusually difficult. On a multijointed, anthropomorphic robot arm, the influence of each joint variable on the resulting position and orientation of the robot's tool surpasses human intuition, so robotics has a special need for graphic as well as mathematical tools.

Here we focus on kinematics. All three of these areas, however, require mathematics, especially analytical geometry; the techniques of this work could be extended to produce computer tools for the analysis of robot control and planning problems as well.

1.3 Current Tools

Robotics is inherently interdisciplinary. Robotic systems, and computer-coordinated mechanical systems in general, involve many technologies, and the level of interaction of mechanical, electronic, and computational elements is

high. For this reason, the designers of such systems need varied computer-aided design and computer-aided engineering tools, ranging from drafting systems for layout of manufacturing drawings, through a variety of simulation and visualization tools, to control-system and mechanism-synthesis tools. "The primary need for these complex systems is for assistance of an integrative nature. Presently, computer-aided engineering tools do not address the integration of complex systems."[Waldron 1987].

Robotics tools in use today range from general and theoretical tools for research and design to specific and rough-and-ready tools for developing and debugging a robot and to sophisticated proprietary CAD/CAM (computer-aided design and manufacturing) systems for automating the development of robotic applications. However, symbolic tools to aid mechanical analysis are largely lacking.

1.3.1 Simulation Tools and Graphic Tools

A robot simulator is a system that accepts commands in the robot's programming language (and perhaps sensory inputs to the robot as well), and provides reports on the resulting state of the robot. Some sort of simulation is essential for debugging a robot; Intelledex, Inc., one of the new wave of robot manufacturers that arose in the early 1980's, found it necessary in its first months of operation, to develop a small microprocessor-based numerical simulator, even though it had not been an item on the first year's agenda. With increasing sophistication, simulation becomes an important productivity factor, allowing

some of the development of robot applications to proceed off-line, without tying up a robot.

One way to report the state of the robot is by graphic display of the robot arm in space. The complicated, three-dimensional nature of the motions of many robotic systems makes the use of computer-aided visualization techniques extremely important, as Waldron points out [Waldron, 1987].

Methods of graphic representation include wire-frame modeling (showing all edges of a body as lines), surface modeling and solid modeling.

Graphic simulators run on various sizes of computer and offer various degrees of simulation. Some examples are Unigraphics' PLACE [Stauffer, 1984], the robotics module in CATIA CAD/CAM [Dombre, Borrel, and Liegeois, 1984], Computervision's ROBOGRAPHICS [Stauffer, 1984] and more recent CADDStation system, Calma's ROBOTSIM [Stauffer, 1984], GMSolid's ROBOTEACH [Pickett, Tilove, Shapiro, 1983], GDP's Emula [Meyer, 1981], and University of Tokyo's GEOMAP [Sata, Kimura, and Amano, 1981]. Silma produces a workstation-based graphic simulator called CimStation, which they program to order, for modeling the operating specifications of a customer's particular system of interactive, synchronously-operating robots and other equipment. It represents devices and workcell as colored wire-frame images.

The state of the art is changing rapidly. The most current information on proprietary robotics tools is found in trade publications like *CIME* (Computers In Mechanical Engineering) and *Robotics Today*.

1.3.2 Mathematical Tools

The kinematic, control and coordination software of advanced robots involves the use of complex algorithms based on quite complicated sets of algebraic and trigonometrical equations. The use of computer algebra programs such as MACSYMA and REDUCE to generate necessary expressions has become an important part of the software design for industrial robots in some quarters [Waldron 1987]. However, there are few if any programs available today that integrate mathematical tools with other robotics tools.

The novelty of RobotScribe lies in two aspects: first, that it uses computer algebra to provide ready-made tools for generating the kinematic equations of a robot arm, and second, that it integrates this facility with simulation and graphic-representation tools in an interactive, windows-oriented environment.

Chapter 2

Robot Arm Kinematics

The first step in our project is to design a simple, concise, yet sufficiently general, representation of robot arm links. In this chapter we formalize kinematic concepts and investigate the problems of constructing mathematical models of the robot arm and other objects in space that lead to usable computer representations. We define "robot" and establish the point of view that this paper takes along the spectrum from theoretical to applied, and among the various disciplines that pertain to robotics. We develop some vocabulary for kinematic aspects of robotic arms that we use later to describe the application and use of our tool. We discuss some of the conceptual and practical questions involved in understanding and making effective use of a particular robot arm.

2.1 Definitions of a "Robot"

Since Karel Capek gave us the word in his 1921 play "R.U.R" [Capek, 1921], and more so since Issac Asimov formulated his "Three Laws of Robotics" [Asimov, 1942], "robot" has suggested futurism and mythology as well as industry and mechanism. Today both aspects are represented in research.

The futuristic view, conceiving of a robot as an anthropomorphic machine approaching human powers, is expressed today in the field of Artificial Intelligence. Indeed Patrick Winston, addressing the 1987 AAAI convention, announced, "We used to consider robotics a branch of AI. I now believe that AI is rather a branch of robotics." [Winston, 1987].

The industrial point of view is a more modest one, directed toward the practicalities of industrial application. Jim T. Dallam, a project engineer with Cincinnati Milacron, says in UPCAEDM, "A robot is really just another machine. In fact a lot of people have trouble drawing a distinction between a robot and an NC [numerically controlled] machine tool." [Dallam, 1987].

The Robot Institute of America (now the Robotic Industries Association) took this tack in their 1979 pronouncement: "A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks." [Weiland, 1985].

This project focuses on current, existing and practical robotic systems - or more generally - computer-coordinated mechanical systems, rather than future, hypothetical, or AI-lab robots.

More specifically, we define a robot arm to be an open sequence of actuated joints with rigid members between. From this mechanical abstraction of the actual robot arm we will proceed toward a mathematical abstraction of the mechanical robot arm in a form amenable to analysis and representation.

The serial linkage geometry of robot arms is described by systems of nonlinear equations. Effective analysis is necessary to characterize the geometric and kinematic behavior of the arm. "This represents an important and unique area of robotics research, since research in kinematics and design has

traditionally focused upon single-input mechanisms, with single actuators moving at constant speeds, while robots are multi-input spatial mechanisms which require more sophisticated tools." [Asada and Slotine, 1987].

2.2 Describing the Links of a Robot Arm

Here we are regarding the robot manipulator arm in its mechanical aspect. Our goal is to define the mathematical relationship between the set of values of the joint variables on the one hand and the position and orientation of the end-effector on the other.

We view the robot arm as a sequence of rigid links, each connected to the previous link at its joint. At each joint, the following link is able to rotate about or to translate along the joint axis. (Multidimensional joints are not allowed, but their effects can be represented by a sequence of single-axis joints; e.g., a spherical, or ball-and-socket, joint can be represented as three rotations about orthogonal axes.)

The kinematic chain begins with a base link, which is often affixed to the ground and is in any case at rest with respect to the external frame of reference; the last link is the end-effector, or "tool", whose position the robot arm is intended to control. The effective point of a robot is a point at the outboard, or distal, end of the end-effector, which is considered the robot's point of contact with the workpiece. As one travels along the arm, the inboard direction is toward the base, and the outboard direction is toward the end-effector.

A link is revolute if its joint variable is angular, that is, if it rotates about its axis. A link is prismatic if its variable is linear, that is, if it telescopes or slides along its axis. A link is fixed if neither its angle nor its length is variable. Some authors define screw, cylindric, planar, and spherical links, and so on [Featherstone, 1987], but all of these types can be expressed as (and often are physically realized as) a succession of two or three prismatic or revolute links, possibly with some constraint on their motion; for instance a screw can be expressed as a prismatic link followed by a revolute link with the constraint that the distance traveled along the prismatic axis is a fixed multiple of the angle opened about the revolute axis.

The customary method of analyzing a robot arm kinematically is to describe each successive link relative to a coordinate system embedded in the previous link. This approach lets us pin down all aspects of a link, except its joint variable, in invariant parameters even though all coordinates of a link are transformed as the arm moves by the effects of the links inboard of it.

The chief issues in the choice of formalism for link description are human intelligibility, generality, and compatibility with the most frequent calculations, which are mainly in the derivation of the transformation of end-effector position due to inserting that link in the arm.

The representation of robot arm links most prevalent in the literature is the Denavit-Hartenberg representation, made popular by [Paul, 1981]. This representation allows a link to have one joint, about one axis only, but it is quite

general otherwise, in that it allows any sort of twist and displacement along the link. Therefore, successive joints can have axes in arbitrary directions.

In order to specify a canonical relative coordinate system for in each link, this method requires that we find the common normal line segments of successive link axes. (Note that any two lines in three- dimensional space have a unique line segment that intersects them both and is normal, i.e., perpendicular, to both of them, except when the two lines are coplanar. Coplanar lines are either parallel, in which case they have infinitely many common normal line segments, all of the same length and parallel to each other, or intersecting, in which case the common normal degenerates to a point, namely the point of intersection.) The coordinate system of Link i-1 then is defined to have its x-axis along the common normal of the previous link axis and this link axis, and its z-axis along this link axis; the y-axis is determined by requiring a right-handed coordinate system. The coordinate system for Link i is similarly defined with z-axis along the common normal of the axes of Link i and Link i+1 and x-axis along the axis of Link i+1. Having established these relative coordinate systems, Link i can be described by two distances and two angles, as shown in Figure 2-1.

Thus the Denavit-Hartenberg representation describes each link by four real numbers, of which one is the joint variable and the other three are link parameters. The translation and rotation due to that link can be expressed as multiplication by the 4 x 4 homogeneous transformation in Figure 2-2.

This link formalism has considerable generality. Its disadvantage lies in being less than transparent, even to sophisticated users. One fully arbitrary link

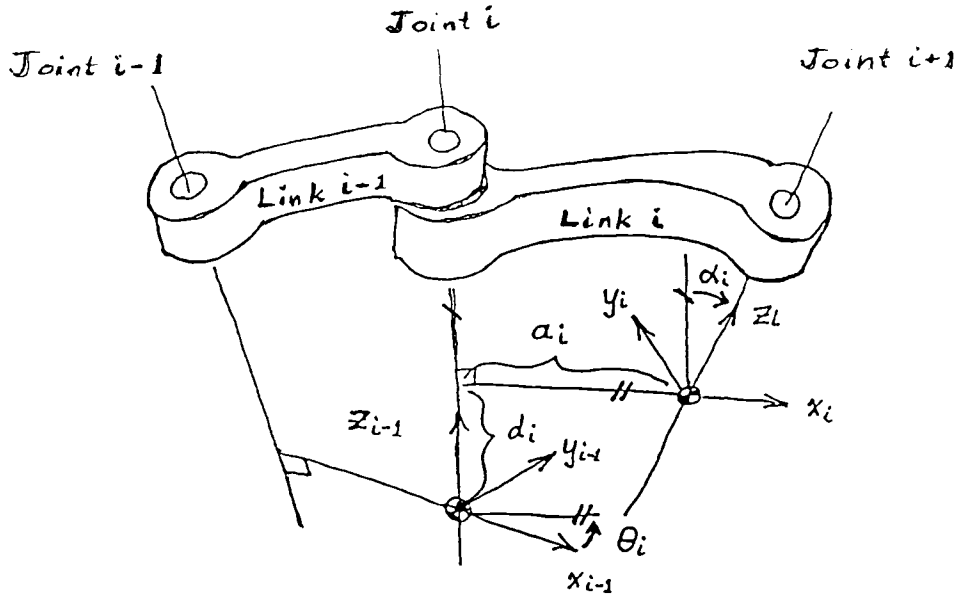


Figure 2-1 General link

$$(A_n) \leftarrow \begin{bmatrix} \cos(\theta) & -\cos(\alpha)\cdot\sin(\theta) & \sin(\alpha)\cdot\sin(\theta) & \cos(\theta)\cdot a \\ \sin(\theta) & \cos(\alpha)\cdot\cos(\theta) & -\cos(\theta)\cdot\sin(\alpha) & \sin(\theta)\cdot a \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2-2 Homogeneous transformation for link

moving in three-space can be comprehended, more or less, but a chain of half a dozen arbitrary links boggles the mind. In practice, real industrial robot arms usually have each successive link axis parallel or perpendicular to the previous link axis.

For this project we propose a simpler representation of robot arm links, which we call orthogonal representation. We restrict the direction of each link axis to be one of the three orthogonal directions defined by the previous link's coordinate system.

Our link description formalism is to describe the link as revolute, prismatic or fixed; to name its axis direction as "x", "y", or "z"; and to give its fixed length, in the case of a revolute link, or its fixed twist, in the case of a prismatic link, or both, in the case of a fixed link. Thus, instead of the four real numbers needed to describe a link in the Denavit-Hartenberg representation, we describe a link using two real numbers, a choice among three directions, and a choice among three types (revolute, primatic or fixed). See Figure 2-3.

Full generality is restored (in case one might want to have link axes taking off in arbitrary directions) by allowing "fixed" links that have no joint variable but that insert a fixed angular twist before the next link. A link whose axis is not orthogonal to the previous link can be represented by combining up to three links, of which the first two are fixed links.

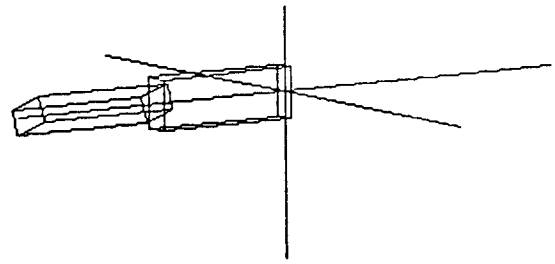


Figure 2-3a Orthogonal link i: revolute(x,d)

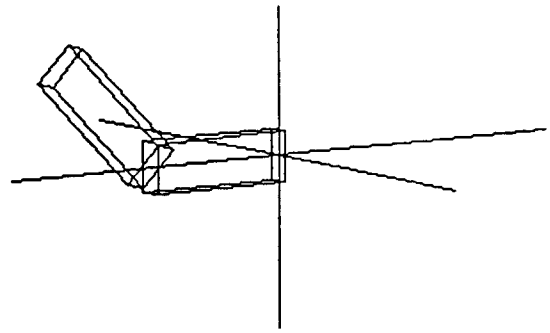


Figure 2-3b Orthogonal link i: revolute(y,d)

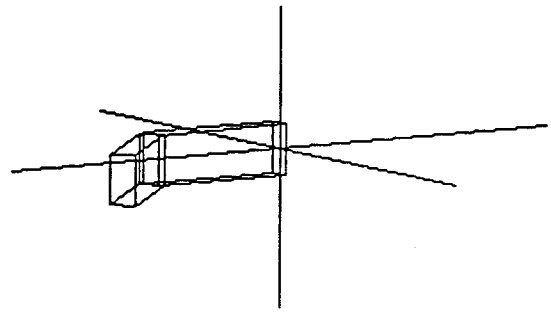


Figure 2-3c Orthogonal link i: revolute(z,d)

2.3 Describing A Rigid Body in Three-Space

A rigid body has orientation (how it is turned in space) as well as location (where it lies in space), which amounts to six degrees of freedom altogether. Rotation is change in orientation; translation is change in location. When we speak of the position of a rigid body we mean its location and orientation considered together. The chief issues in choosing a mathematical system for describing the position of a rigid body in space are intelligibility to human users (on the shop floor, for instance), conciseness, and compatibility with required calculations.

The location of a point in space is determined by 3 coordinates. Rectangular coordinates (three orthogonal distances) are the most commonly used; cylindrical coordinates (an angle, a radial distance and a perpendicular distance) and spherical coordinates (a radial distance and two angles) are used also. We use rectangular coordinates here. The location of the rigid body is then specified by the coordinates of an agreed-on point in the body.

There are a number of systems for specifying orientation. Some of them are based on the idea of comparing the given orientation of the body to some standard initial orientation, and describing the rotation, or series of rotations, needed to transform the initial orientation into the given orientation. We assume that the standard initial orientation of the body is lying with its principal axis along the positive x axis.

2.3.1 Matrix Systems

One system that is prevalent in robotics literature today, due to the influence of R.P.Paul's seminal book *Robot Manipulators: Mathematics, Programming and Control* [Paul, 1981], is to describe an orientation by giving the 3 x 3 rotation matrix as in Figure 2-4 that would transform the body from its initial orientation into a given orientation. Advantages of this system are that the rotation matrix arises naturally in the forward kinematic derivation (see Section 2.4.1), and it can be combined with a location vector to form a 4 x 4 homogeneous transformation [Denavit and Hartenberg, 1955], describing both location and orientation, as in Figure 2-5.

This system has some drawbacks, however. Since there are actually only three degrees of freedom for the orientation of a rigid body, describing it by nine real numbers not only unnecessarily expands the amount of information one must carry, but also makes it very cumbersome to specify an orientation a priori - the nine numbers must satisfy consistency constraints, which lead to additional calculations. As Paul says, "These restrictions ... make it difficult to assign components to the vectors except in simple cases when the end effector is aligned with the coordinate axes" [Paul, 1981].

The 4 x 4 homogeneous form brings in further verbosity; besides the nine rotational coefficients and the three location coordinates, it requires four dummy coefficients to round out the form. The apparent advantage of compactness in putting rotation and location information into one structure is often outweighed

$$\begin{bmatrix} nx & ox & ax \\ ny & oy & ay \\ nz & oz & az \end{bmatrix}$$

Figure 2-4 Orientation matrix

$$\begin{bmatrix} nx & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2-5 Homogeneous transformation

by the clumsiness of carrying sixteen coefficients to express six degrees of freedom.

2.3.2 Successive Rotation Systems

Other systems for describing orientation, in use in astronomy and in navigation for quite some time, describe orientation in terms of a sequence of rotations about specified axes. Note that the order in which the rotations are made is significant, since rotations in three-space are not commutative.

Euler angles describe any possible orientation by giving three angles of successive rotations about orthogonal axes relative to the rigid body: first a rotation about the z axis, followed by a rotation about the new x axis, and finally a rotation about the new z axis [Asada and Slotine, 1986].

Yaw, pitch and roll are a series of three angles commonly used in describing orientation of an airplane. Yaw is the rotation about the vertical axis; pitch is vertical declination, which is rotation about the new y axis; and roll is a rotation about the principal axis of the body, which is the new x axis. In robotics, it is common to make the positive sense of pitch be downward, since a robotic arm generally has its end-effector pointing downward toward a workpiece located below it. See Figure 2-6.

These two systems score high for conciseness and reasonably high for intelligibility. They both suffer from the possibility of degeneracy; certain values of the middle angle in the sequence destroy the independence of the first

and third angles. When pitch is plus or minus pi/2, yaw and roll coincide so the description is not uniquely defined.

2.3.3 Two-Point or Point-and-Vector Systems

Another approach is to think of a rigid body as being brought into position in space by first pinning down one reference point in the body, then pinning down a second point in the body, distinct from the first point, and finally specifying rotation of the body about the line between the two points. In this approach, location is determined as before by the coordinates of the first point and two dimensions of orientation are determined by the second point (more precisely, by the line from the second to the first point).

We specify the location of the end-effector by giving the coordinates of its effective point. We can specify the orientation of the end-effector by giving the location of the wrist, at the inboard end of the end-effector, plus the roll of the end-effector about its principal axis. This method has high intelligibility and no degeneracy problem. It is not as concise as possible, since it requires seven real variables, namely the three coordinates of the effective point, the three coordinates of the wrist location, and the angle of roll about the line from wrist to effective point; any one of the three coordinates of the inboard end can be calculated from the other two, given the fixed length of the end-effector and the location of its outboard end.

A variation on this idea is to specify orientation by giving the first two coordinates of a unit vector along the principal axis of the end-effector, along

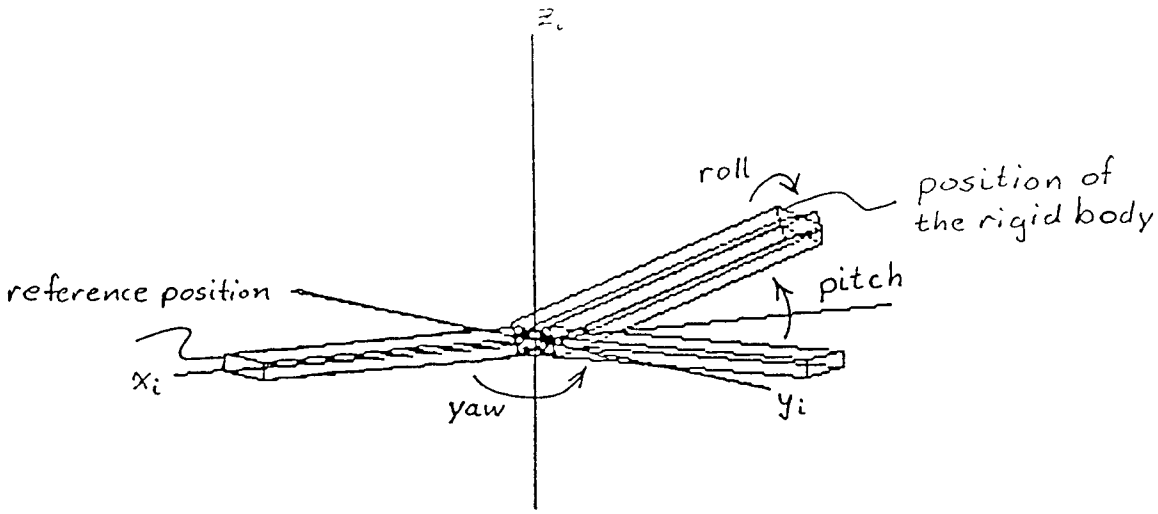


Figure 2-6 Successive-angle orientation description

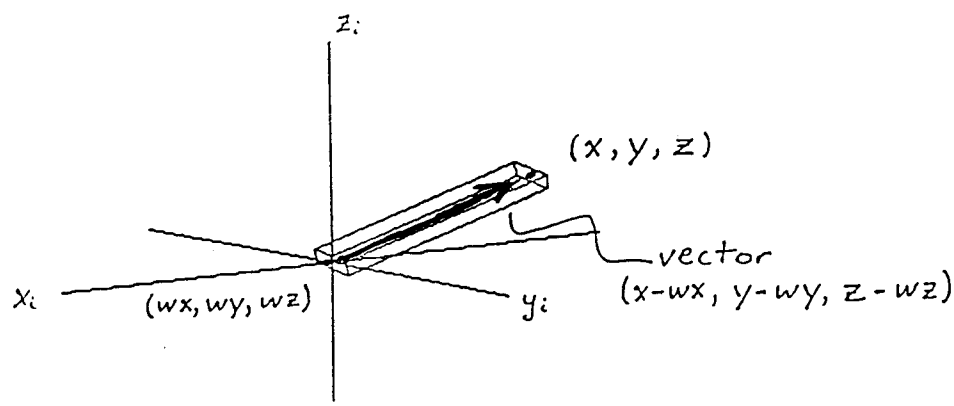


Figure 2-7 Point-and-vector orientation description

with the roll about that axis. See Figure 2-7. A unit vector is not quite as concrete to the average person as an endpoint, but by cutting the parameters down to six, we eliminate the problems of dependency, as well as gaining conciseness.

Both of these systems are free of degeneracy. They are both reasonably apt for expressing a priori descriptions.

Of the methods discussed here, we find matrix representation of orientation too unwieldy. The yaw-pitch-and-roll system is common in industrial robot programming languages like Intelledex's RBASIC [Intelledex, 1983], and is the system used in the rest of this work. The point-and-vector system may be more suitable for analytic work.

2.4 Transformations between Joint Space and World Coordinate Space

A robot's task is specified in a frame of reference called the work cell, and generally involves various objects located within the work cell which the robot is to touch, grasp, move, sense, weld, spray, or otherwise manipulate. To perform the task requires the robot to bring its end-effector, through movement of the robot arm linkages, to an appropriate point in space in an appropriate orientation. Location and orientation relative to the work cell are described by world coordinates.

The position of the end-effector is controlled by the values of its joint variables. The joint space of a given robot is the space of possible joint values; its dimension is the number of links in the robot arm.

2.4.1 The Forward Kinematic Equations: Joint to World

To find the coordinates of the end-effector as a rigid body in space, given the values of the joint variables, is the forward kinematic problem. The resulting Robot Equation answers the practical question, "Where in space is the end-effector now?" given that we know the state of the robot's joints.

The method of solving the forward kinematic problem is to express for each link the transformation of the position of the end-effector that is due to the presence of that link in the arm, and then to find the composition of these transformations from the base to the last link outboard. A revolute joint produces a rotation depending on its joint variable, and a translation also if the length is nonzero; a prismatic joint produces a translation depending on its joint variable, and also a rotation, if its twist angle is nonzero. The solution is straightforward, though rather tedious, to perform by hand.

The transformations at each step can be represented in several different ways. If one uses 4 x 4 homogeneous matrices to represent both the translation and the rotation due to a link, then the composition of the effects of all the links is found as the matrix product of each link's homogeneous matrix, from the base to the end-effector.

A more direct method, avoiding the artifice of 4 x 4 matrices, is to compute, for each link i, a rotation matrix M(i), a cumulative rotation matrix RM(i), and a cumulative translation vector P(i) together, by the relations shown in Figure 2-8.

2.4.2 The Inverse Kinematic Equations: World to Joint

The inverse kinematic problem is to find the set of joint values required to achieve desired spatial coordinates for the location and orientation of the end-effector. The Inverse Robot Equation answers the question, "By what joint values can a desired location and orientation of the end-effector be achieved?" There is no general method for deriving the Inverse Equations; in fact there are configurations for which no analytic solution exists.

Note that, although the forward equations can be expressed as a matrix equation, when revolute joints are involved the equations are far from being linear in the joint variables, since they involve trigonometric functions of the joint variables. A purely algebraic attack tends to give rise to fourth degree equations. However, Pieper has shown [Pieper, 1968] that a wide class of configurations do have analytic solutions, namely those where the problem can be partitioned into subproblems each involving three or fewer joints.

In practice, designers derive Inverse Equations for their robot arm configurations by a combination of analytic technique, geometric insight and trial-and-error. Even more than the forward direction, finding the Inverse Robot Equation can involve extended manipulation of matrices with rational

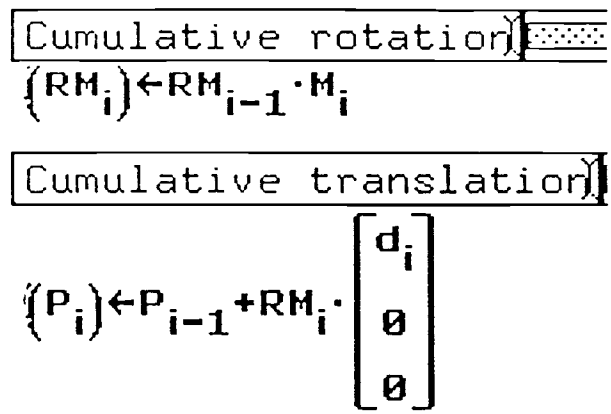


Figure 2-8 Cumulative rotation and translation

expressions of trigonometric expressions as matrix elements. This makes it a good application for computer algebra.

2.5 Configurations

The configuration of a robot arm is determined by the type and direction of its joints and their sequential order. Configuration determines functional properties of the arm such as dexterity, reach and work envelope. Configuration also determines the kinematic properties of the arm.

A configuration is called "universal" if it has at least six degrees of freedom, so arranged as to allow the end-effector to be positioned freely at any values of position and orientation that are within arm's reach. To be more precise, an arm is universal if it is capable of continuously varying all six world coordinates of end-effector position at any position interior to the arm's work envelope.

Other cases are:

Insufficient axes. With fewer than six independent joints, the motion of the end-effector is constrained in some of its six coordinates of position and orientation. For example, the five- axis manipulator in Figure 2-9 can position its end-effector at any point x, y, z, and orient it as to pitch and roll, but yaw will always be Arctan2(y, x), that is, pointing radially out from the center. The three-axis arm (Figure 2-10) can position its end-effector in x, y, and z only.

Redundant axes. A manipulator may have joints whose effects coincide. In this case, the joints provide fewer degrees of freedom than the number of joints.

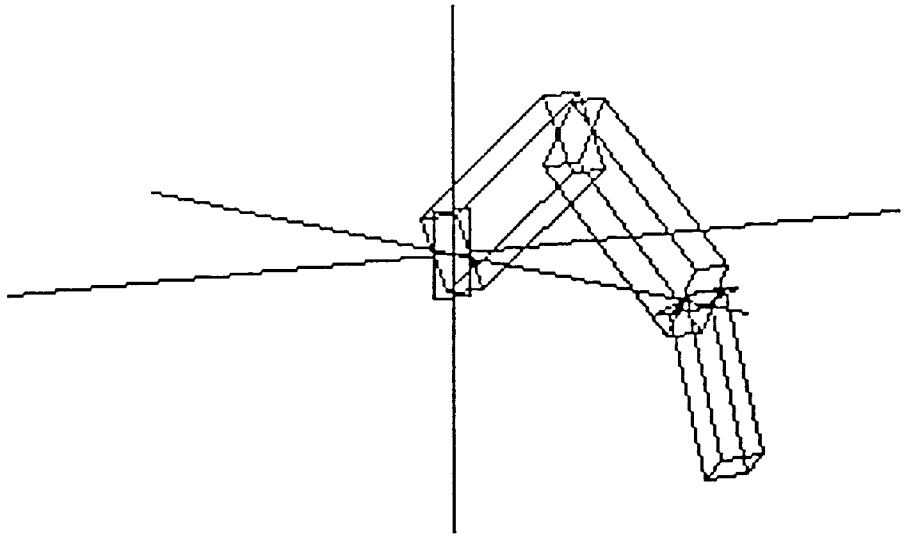


Figure 2-9 A five-link arm

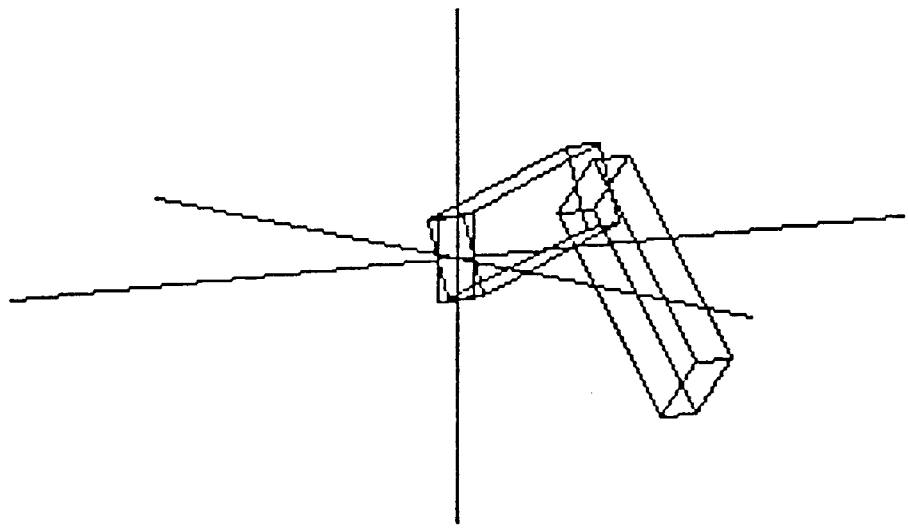


Figure 2-10 A three-link arm

Any number of axes greater than six must have redundancy with respect to spatial positioning. The human arm, for example, has essentially seven axes of motion; this is why it is possible to hold one's hand in a fixed position and still move one's joints continuously so that one's elbow moves around in space.

Degenerate axes. A joint may become locally redundant with another when a third, intervening, joint assumes a certain value. For example, in a sequence of three revolute joint where the first two have zero length (i.e., the three joint axes intersect), certain positions of the middle joint may cause the first and the third joint axes to coincide. Then Joint 1 and Joint 3 have the same action and locally there are only two degrees of freedom.

If a robot arm configuration has fewer joints than there are degrees of freedom in space to satisfy, the arm must be insufficient and the end-effector will be able to attain only a subspace of possible positions in space. If a configuration has more joints than there are degrees of freedom to satisfy, it must be redundant and a given position will be attainable in infinitely many ways. When the number of joints equals the number of degrees of freedom, the arm still may not be universal; if some joint duplicates the effect of another joint, the arm will be degenerate and have both redundancy and insufficiency and the subspace of attainable positions will be attainable in infinitely many ways. Finally, if the number of joints equals the number of degrees of freedom in space with no degeneracy, then positions within the work envelope will generally be attainable in finitely many ways.

Note that in this last, well-behaved, case, a given position still may be attainable in more than one way, if some of the joints are revolute, due to the periodicity and the symmetries of the trigonometric functions which describe revolute motion. This is called ambiguity. These multiple solutions differ by such physical characteristics as elbow-up versus elbow-down, or shoulder-left versus shoulder-right. For kinematic models with closed form inverses, a parameter to specify the unique characteristics of a desired solution can be incorporated into the algorithm.

Chapter 3

An Integrated Environment for Computer Algebra

3.1 Computer Algebra

In this chapter we describe the window-oriented computer-algebra system in which our set of robot tools is implemented.

3.1.1 What Symbolic Computation Is

We program computers in algebraic language all the time. In any high level computer language we use expressions like

```
s := x + 1;
t := x - 3;
write s*t;
```

But in the usual programming language environment, the computer must substitute numerical values for variables before interpreting such expressions. Given a numerical value for x, say x = 6, the computer can use the expression as its algorithm for numerical operations leading to the values s = 7 and t = 3, but it cannot manipulate x + 1 or x - 3 as algebraic expressions, independent of specific numerical values.

That is a pity, one might think, if one has occasion very often to do long algebraic derivations by hand, for most of such work is as tedious and algorithmically determined in algebraic manipulations as in numerical

calculations. Robot designers, for instance, are faced with kinematic derivations that may run to many pages. In contrast, a symbolic computation system can manipulate mathematical expressions. In our example, a computer algebra system can compute

$$s*t = x^2 - 2*x - 3$$

without reference to numerical values.

Computer algebra is the branch of computer science which analyzes and implements algebraic algorithms. See [Buchberger et al., 1982] for further discussion.

3.1.2 The Development of Computer Algebra

In the late 1960s, a number of symbolic computation systems emerged, including Macsyma at MIT and REDUCE at the University of Utah. Each offered a well-defined, structured method for high-level mathematical calculation. At the time, however, only large mainframes could provide the amount of memory they required, so these tools were generally limited to a few university research groups. The system reported here is based on REDUCE.

The capabilities of REDUCE include:

1) expansion and ordering of terms for polynomials and rationals,
2) substitution and pattern matching,
3) several modes for simplification, controlled by switches,
4) calculations with symbolic matrices,
5) arbitrary precision integer and real arithmetic,
6) ability to define new functions and extend program syntax,
7) analytic differentiation and anti-differentiation,
8) factorization of polynomials.

3.1.3 Limitations of the First Generation

For many years, the high memory demands of computer algebra effectively limited systems to large mainframe computers. With the advent of cheap memory it has become feasible to implement a symbolic computation system on a personal workstation. When transferred, however, a system like REDUCE shows its age; the Fortran-like linear format required to input mathematical expressions reveals a crucial limitation: a primitive and awkward interface.

In addition to a state-of-the-art interface, the modern user expects to be able to integrate with other computing modes, especially to have access to libraries of mathematical expressions and to be able to plot expressions.

The MathScribe environment developed at Tektronix Laboratories provides these facilities for the REDUCE computer algebra system. [Smith and Soiffer, 1986].

3.2 The MathScribe Environment

MathScribe is a windows-based environment for symbolic computation with REDUCE as its computer-algebra engine. It combines symbolic, numerical, and graphic facilities into one visually-oriented environment that runs under the UNIX operating system on a Sun workstation.

MathScribe is based on the X Windows screen management conventions that are becoming a standard for visually-oriented interfaces. It runs processes in multiple windows that can be moved about on the screen, scrolled, or collapsed into icons.

MathScribe has mouse-controlled cursors; mouse buttons to select and highlight a region within a window, to pull up menus and to select actions; visual editing tools to add to, delete, copy and cut-and-paste selected regions. Figure 3-1 gives a typical screen layout.

In addition to the usual windows management facilities, MathScribe has extended and enhanced features especially for dealing with mathematical expressions.

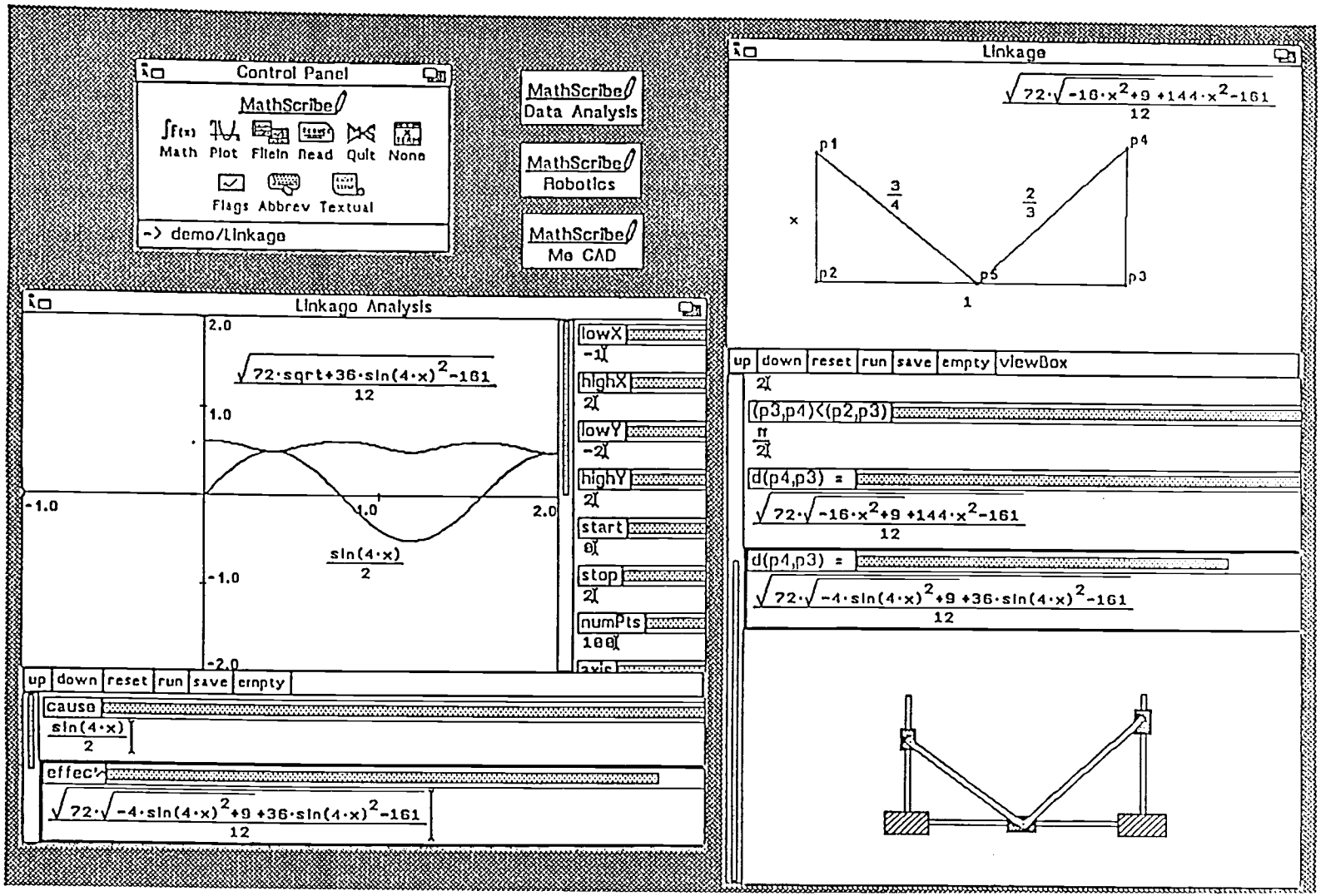


Figure 3-1 MathScribe

MathScribe provides two-dimensional entry and display of mathematical expressions. As the user types from the keyboard in the linear form typical of computer languages, MathScribe constructs its display of the expression in the two-dimensional form used in handwritten and typeset mathematics.

The intelligent cursor follows the grammar of the mathematical syntax of the expression. For instance, in entering a rational expression, when the user types "/" indicating division, MathScribe draws a fraction bar under the numerator and moves the cursor below it, ready to receive the denominator.

Built-in templates for most common mathematical structures simplify entry of information, and templates for all programming structures in the REDUCE language make it possible to construct programs without prior familiarity with REDUCE.

Modern workstations have fast, high-resolution bit-mapped display, providing good capabilities for graphics. MathScribe makes use of these graphics capabilities to provide plotting for functions and relations in two and three dimensions. Animation is provided by a buffer for storing frames that can then be replayed. In this work, we adapted the three-dimensional plotting of MathScribe for the visual representation of robot arms.

Chapter 4

The Parts of RobotScribe

RobotScribe is a set of tools for robot kinematics. Its most significant aspect is the application of computer algebra to kinematics to generate and analyze the kinematic equations of an arbitrary robot in symbolic form. Three-dimensionsal kinematic structures with several joints require aids to visualization; our system provides graphic representations of the robot arm and simulation facilities to show the robot arm as it moves.

RobotScribe consists of four modules. The Kinematics Helper module builds a given robot from a sequence of link descriptions and produces the forward kinematic equations for that robot in algebraic form. The Simulator module imitates typical robot commands including moving the robot arm around. The Drawrobot module plots a three-dimensional wire-frame picture of the robot arm. The Inverse Toolkit module provides a kit of subroutines from which the analytical solution to the Inverse Problem can be built up for many robot arms.

Operations in Kinematics Helper initializes the global data structures and provides the kinematic apparatus which the other three modules use. Operations in the Simulator instantiate the joint variables; these must have numerical values before a picture can be drawn by Drawrobot.

4.1 Kinematics Helper

Recall that the second step in our project is to design and implement, using orthogonal representation, a module to automatically generate the Forward Kinematic Equation in algebraic form for arbitrary robot arm configurations.

The Kinematics Helper module sets up a global data structure for fixed robot parameters; accepts user link descriptions and initializes robot parameters accordingly; generates a rotation matrix for each link; finds cumulative rotation matrices; and calculates the position of each link in the arm with respect to a world coordinate system whose origin is at the beginning of the robot arm.

Note that the routines in this module operate in a computer algebra environment, and their variables can take symbolic as well as numerical values. Therefore, by leaving joint variables uninstantiated, this module can produce the Robot Equations, the system of trigonometric equations that gives position of the end-effector in terms of the joint variables.

In the following sections we describe the commands by which the robot designer directs the Kinematics Helper.

4.1.1 Specifying a Robot Arm

The RobotScribe command

```
initialize();
```

prepares the system to receive a new robot arm description.

The user specifies a robot arm by describing each link in sequence, starting from the base, using the command "revolute" or "prismatic", and supplying parameters to describe the link.

Recall that a variable link is specified by its type (revolute or prismatic), the direction of its axis of motion (x, y, or z), and a number which is either the link's length if the link is revolute, or the link's twist angle if the link is prismatic. Note that our convention is that a link's angle, whether variable or fixed, always occurs inboard of the length. Therefore, the length associated with a revolute link is the length of the following member.

The RobotScribe command

revolute(<whichaxis>, <length>);

defines the next link in the robot arm to be a revolute link which moves about <whichaxis>, relative to the preceding member, and is followed by a member of length <length>. The RobotScribe command

prismatic(<whichaxis>, <twist>);

defines the next link to be a prismatic links which moves along <whichaxis> relative to the preceding member, after being bent through an angle of <twist>.

RobotScribe can accept descriptions with up to twenty links.

When all links have been described, the RobotScribe command

```
generate();
```

causes the system to generate the forward kinematic equations for this robot arm.

The RobotScribe command

```
showeq();
```

displays the expressions for the coordinates of the end effector in terms of the joint variables.

For example, to specify the two-axis manipulator shown in Figure 4-1, we could say:

```
initialize();
 revolute(z,15);
 revolute(z,12);
 generate();
```

Then showeq() would produce the display shown in Figure 4-2.

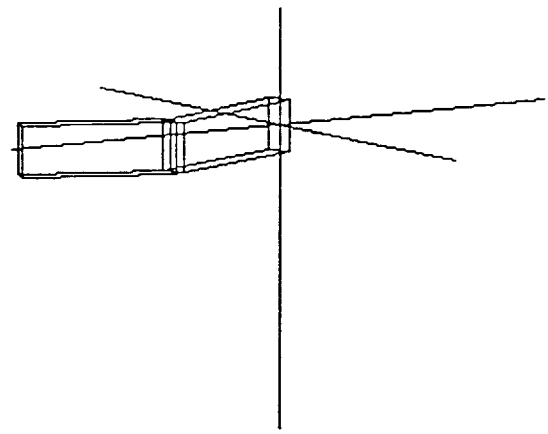


Figure 4-1 A two-link robot arm

```
L30
showeq()
L31
⎡                12·cos(V1)·cos(V2)+15·cos(V1)-12·sin(V1)·sin(V2)⎤
⎢                12·cos(V1)·sin(V2)+12·cos(V2)·sin(V1)+15·sin(V1)⎥
⎢                                                              0⎥
⎣atan2(cos(V1)·sin(V2)+cos(V2)·sin(V1),cos(V1)·cos(V2)-sin(V1)·sin(V2))⎦
                                                               0
```

Figure 4-2 Equations for a two-link arm

A four-link configuration known as SCARA, common in industrial robots, has two revolute links with nonzero length horizontally, a vertical prismatic link, and a fixed link revolute about the principal axis of the tool. To specify the four-axis, SCARA-type robot arm shown in Figure 4-3, we would give these specifications:

```
initialize();
revolute(z,12);
revolute(z,12);
prismatic(y,-pi/2);
revolute(x,8);
generate();
```

and then use showeq() to produce the display shown in Figure 4-4.

Using the capabilities of MathScribe, one can manipulate, simplify, and rearrange these expressions into a more desirable form.

4.1.2 Associated Kinematic Apparatus

In producing the kinematic equations, RobotScribe has generated the cumulative-rotation matrix for each link in the arm. These are represented in a global array of 3x3 matrices called RM. The matrix RM(i) represents the cumulative rotation in following the arm from the base through link i.

For example, the final cumulative rotation on the four-link robot arm of Figure 4-3 is given in RM(4) as shown in Figure 4-5.

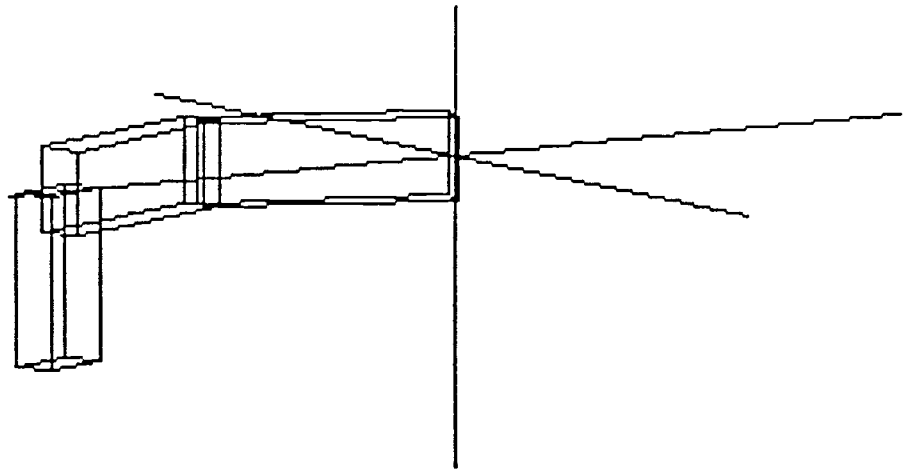


Figure 4-3 A four-link SCARA-type robot arm

```
L0
showeq()
L1
12·cos(V1)·cos(V2)+12·cos(V1)-12·sin(V1)·sin(V2)
12·cos(V1)·sin(V2)+12·cos(V2)·sin(V1)+12·sin(V1)
-V3-8
atan2(0,0)
asin(1)
```

Figure 4-4 Equations for a four-link SCARA-type robot arm

RobotScribe has also generated a position vector for the outboard end of each link, in the form of a global array of 1x3 matrices called P.

Figure 4-6 shows the results of displaying P(1), P(2), P(3), and P(4) for the robot arm of Figure 4-3.

4.2 Simulator and Graphic Display

The third step in our project is to complement the kinematics module with a robot simulator and a graphic display. Because of the extreme spatial complexity that is typical of robot arms, some form of visual simulation is indispensable for keeping track of what is going on.

Once the kinematic framework has been generated for a particular robot, the Simulator module instantiates the joint variables to numerical values. This module contains a number of commands which simulate typical robotics-language commands for motion of the arm.

The RobotScribe command

simulate()

sets up the system for simulation, and instantiates each joint variable to its home position.

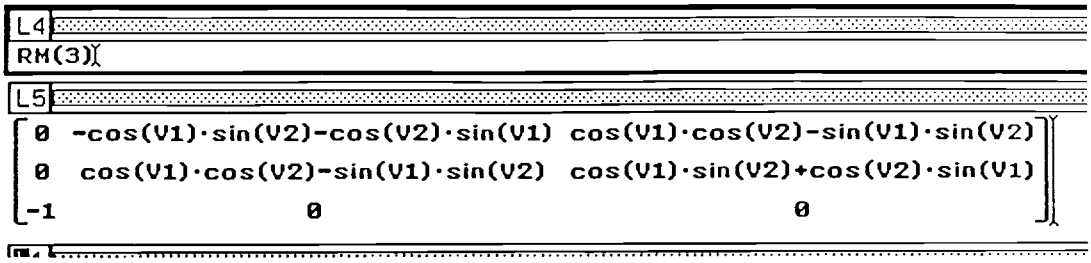


Figure 4-5 Cumulative rotation matrix for SCARA

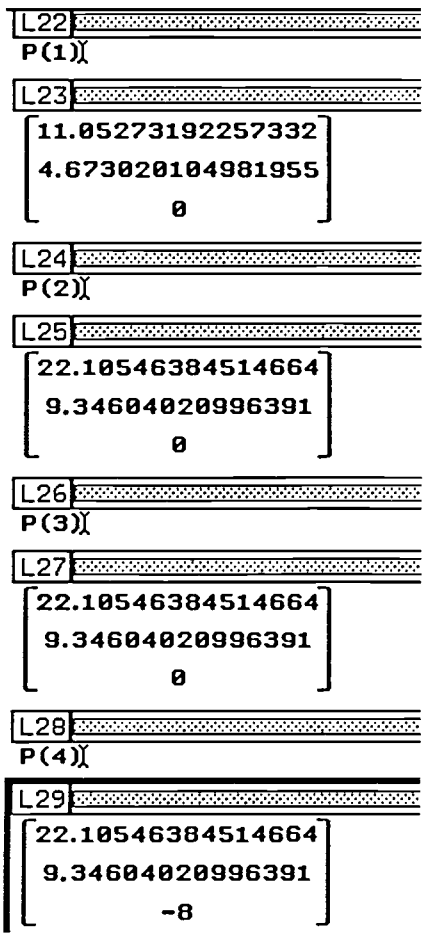


Figure 4-6 Location vectors for SCARA

Jointwise motion can be accomplished by the RobotScribe commands

imove1(<displacement>), ... , imove12(<displacement>)

for incrementally changing the joint variable of links 1 through 12 by displacement. Displacements for revolute joints are in radians. (Prismatic joint displacements are in whatever linear units the user has chosen.)

In Simulate, expressions for kinematic apparatus that work in symbolic form in Kinematics Helper are now instantiated with numeric values. Figure 4-7 shows some examples for a four-axis, SCARA-type arm.

The Drawrobot module brings to bear the plotting capabilities of MathScribe to display a wire-frame representation of a given robot arm in a given position. The specification of the robot arm via Kinematics Helper and the instantiation of joint values via the Simulator must have been done before a representation can be drawn.

To draw anything, one must pull up a MathScribe plotting window and initialize it. We provide a stored plotting window with appropriate dimensions and point of view already set up for drawing a robot.

The various Simulator commands serve to change the values of the joint variables and thus to change the position of the arm (in the internal model).

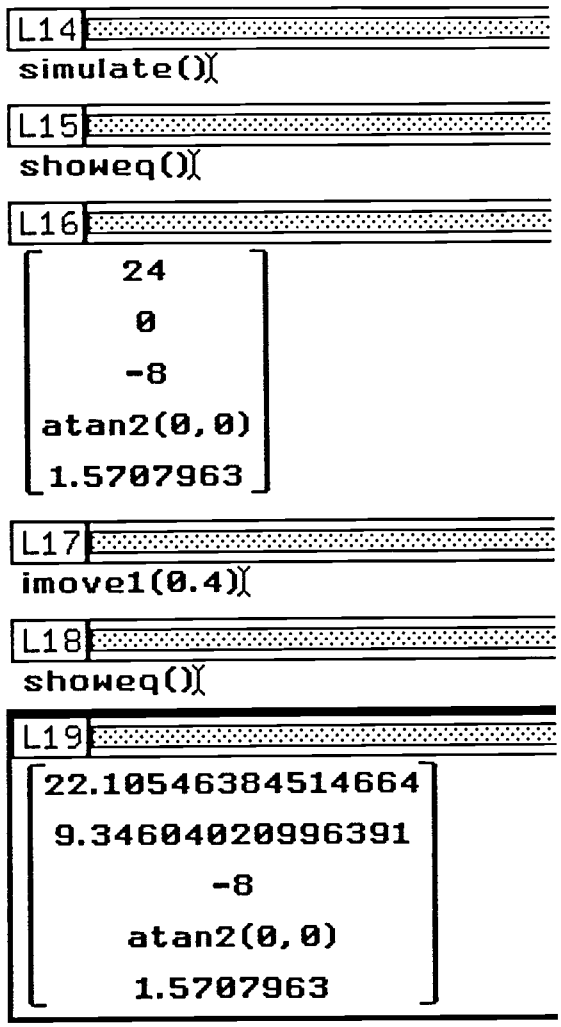


Figure 4-7a Numerical world coordinates

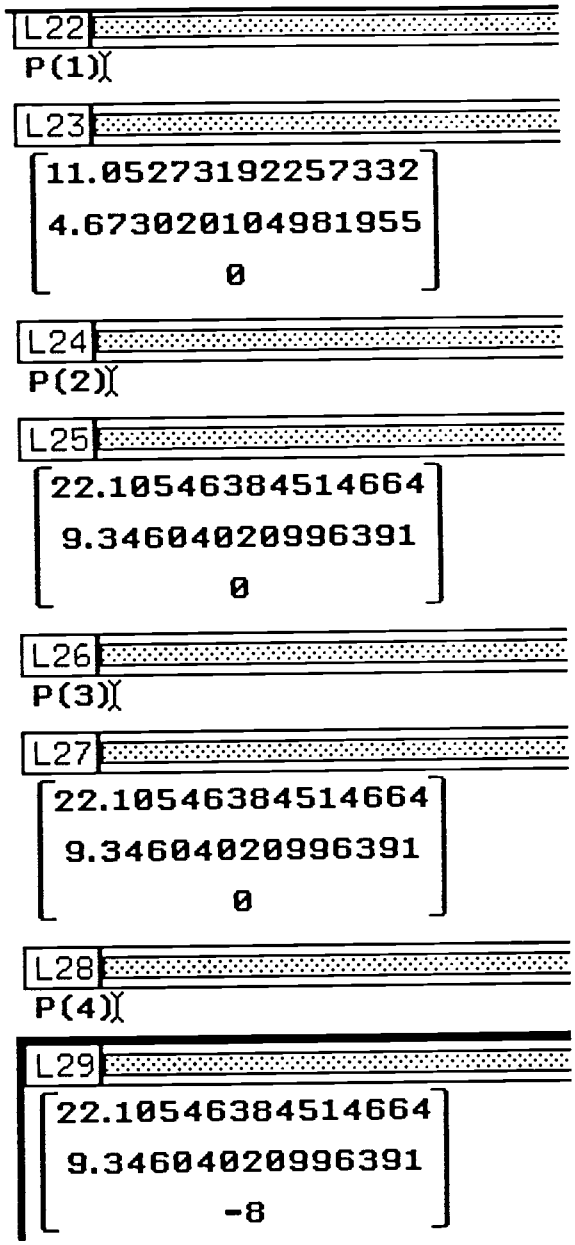


Figure 4-7b Numerical locations

```
L20
RM(3)
L21
⎡ 0  -0.3894183420818296  0.9210609935477768⎤
⎢ 0   0.9210609935477768  0.3894183420818296⎥
⎣-1                 0                   0  ⎦
```

Figure 4-7c Numerical cumulative rotation matrix

When the user selects

drawrobot()

the wire-frame representation of the robot arm in its current position is drawn in the plot window.

Various features of MathScribe allow flexible manipulation of this setup. Parameters can be changed as desired. The input and the output on the screen from RobotScribe routines can be edited and algebraically manipulated, as detailed in the MathScribe User's Manual. [Tektronix, 1988].

The animation capability of MathScribe allows the user to save frames into an animation buffer. If one repeats the cycle

1.) Move the arm
2.) Draw the arm
3.) Save into the animation buffer

one can see the robot arm in motion by replaying the animation.

For example, let us suppose that we have already specified the four-axis SCARA-type robot discussed previously, and now we want to see it. First, we select

simulate()

to instantiate the joint variables at their home position. Next, we select

drawrobot()

to see a wire-frame representation of the arm in its home position as in Figure 4-8a. Select

save

under animation control to make this view the first frame in an animation. Next, let us move the arm around with the incremental joint moves. Select

imove1(0.4)

to move Link 1 by 0.4 radians, and then select

drawrobot()

to see the arm as in Figure 4-8b. Select

save

to make this the second frame of the animation. Now let us see Link 2 move by selecting

imove2(-0.4)

followed by

drawrobot()

which will show us Figure 4-8c. We save it, and then move Link 3 thus

```
imove3(5)
drawrobot()
save
```

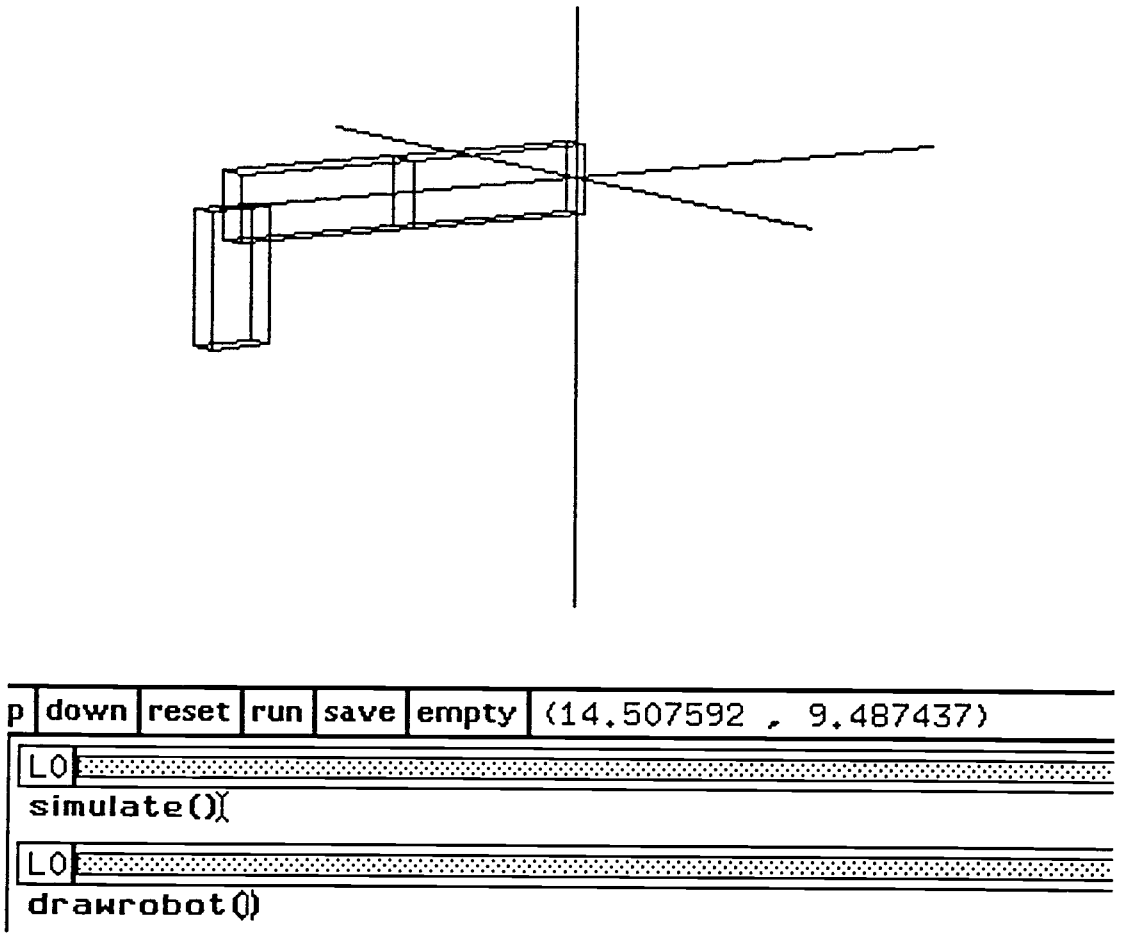
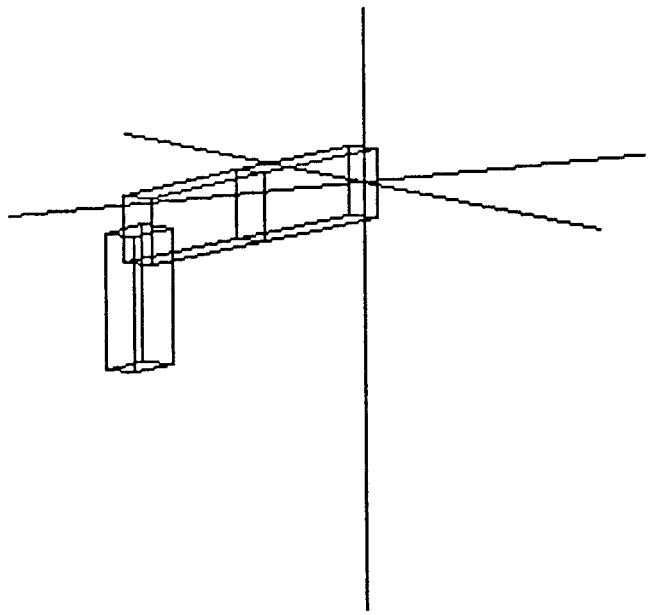


Figure 4-8a Drawing the SCARA



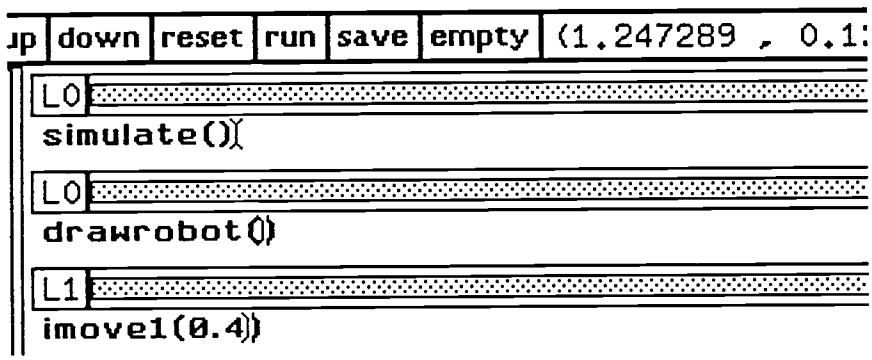
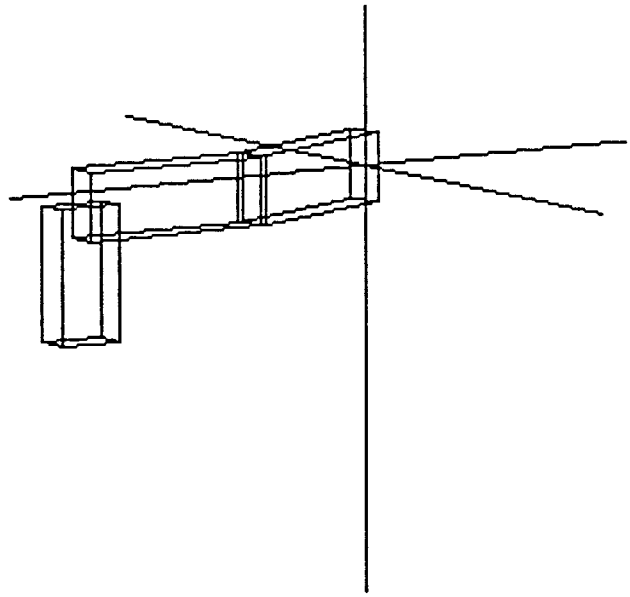


Figure 4-8b Drawing the SCARA



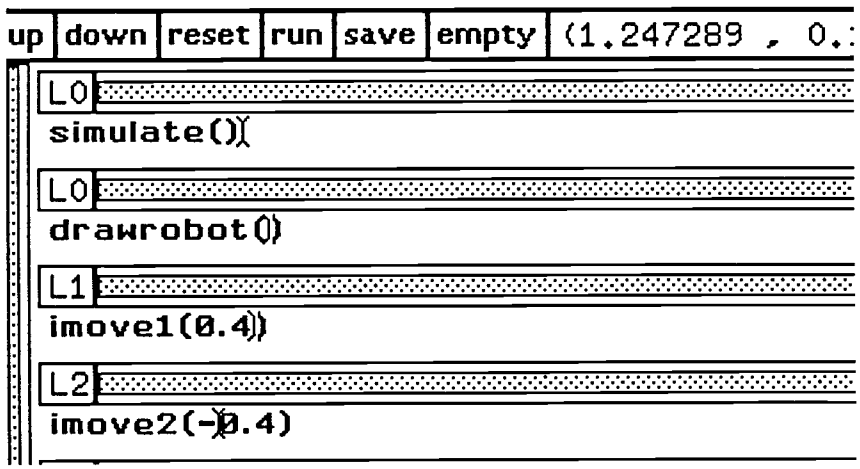
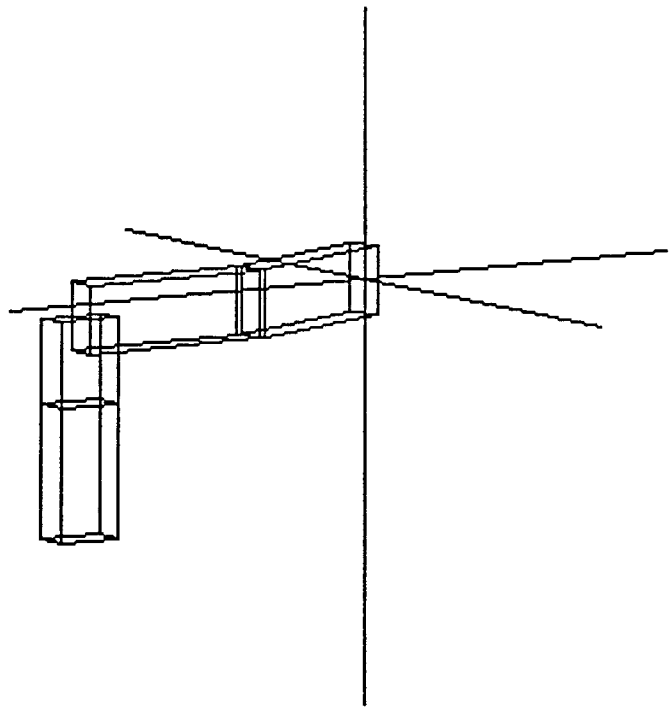


Figure 4-8c Drawing the SCARA



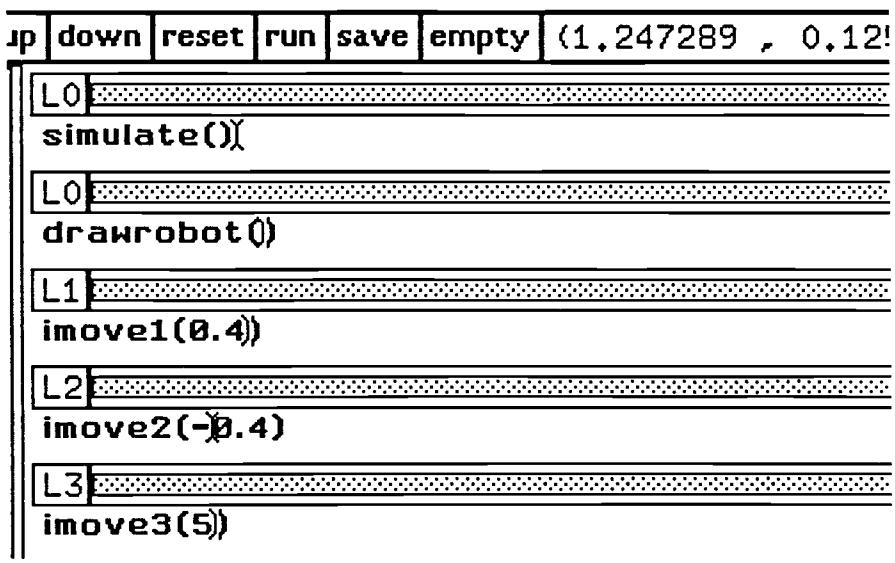


Figure 4-8d Drawing the SCARA

and see the fourth frame, shown in Figure 4-8d.

To play back the animation, select

run

in the animation control area.

4.3 Inverse Kinematic Equations Toolkit

The fourth step in our project is to attack the Inverse Kinematic problem by analyzing the frequently-occurring sub- configurations, and then implementing subsolutions from the entire arm's solution is built.

Recall from Section 2.4.2 that there is no algorithmic method, given the links specifications of a robot arm, leading to the solution of the kinematic equations for the joint variables in terms of world coordinates of the effective point.

For a number of industrial robots, the Inverse Problem responds well to a geometrical approach. With a bit of thought, one can often partition the problem by associating the task of controlling specific world coordinates to a specific sequence of one, two or three consecutive links.

For instance, in the four-axis arm of Figure 4-3, one can see that Link 3, with the prismatic joint, has the role of determining the world z coordinate of the end effector, Link 4 is responsible for world coordinate roll, and Links 1 and 2 together determine the world x and y coordinates of the end-effector.

Certain processes for extracting joint values from world coordinate values recur in many different configurations. When one link entirely controls one coordinate, it is trivial to extract it, e.g., in the four axis robot above, V3 = -z, possibly plus a constant. The relationship between the world x and y coordinates and joint variables V1 and V2 is not as quick to see, but is also a common kinematic transformation step. All elbow-shaped mechanisms where two world coordinates in a plane are determined by two consecutive revolute joint variables with parallel axes and nonzero lengths, give rise to basically the same problem of solving a triangle with three known sides by means of the Laws of Cosines and of Sines.

RobotScribe is not able to provide automatic solution to the inverse problem for arbitrary robots; human intelligence is still necessary for partitioning the problem. The Inverse Toolkit, however, takes care of most of the computational work involved in arriving at the solution.

For each robot arm specified, the user can construct a corresponding version of a REDUCE procedure

inverse({x, y, z, yaw, pitch, roll})

which returns a list

{V1, V2, V3, . . . , Vn}

of expressions for the joint variables in terms of the world coordinates. Because MathScribe is equipped with a menu-driven set of programming templates, it is not necessary for the user to know the syntax of REDUCE.

The Inverse Toolkit is a collection of routines to perform the mathematical calculations for several of these common processes. The elements of the toolbox include the procedures wristposition, shouldertwist, sweep, and elbow.

World coordinates specify the orientation of the end-effector, i.e. the last link, and the position of its outboard end. The location of the inboard end of the end-effector, on the axis of the preceding joint, is often called the "wrist". From the orientation, one may derive a unit vector in the direction of the end-effector; multiplying it by the length of the last link and subtracting from the tool position gives the coordinates of the wrist. The procedure

wristposition({x, y, z, theta, phi})

uses this principle to take a list of end-effector world coordinates and calculate

{wx, wy, wz}

the position of the wrist.

When several lower links lie in a plane (for example, a vertical plane) and one of the upper revolute joints has the function of selecting the plane, the value of that revolute joint is determined by two coordinates (in this example x and y)

that are orthogonal to the plane of the arm. The procedure

sweep(x,y)

takes two coordinates and returns the angle for the upper joint.

A planar elbow-shaped mechanism has two revolute joints that determine two coordinates in a plane. The procedure

elbow({px, py})

takes a list of two coordinates and returns a list

{angle1, angle2}

of two angles.

When the revolute joint that determines yaw is located up at the shoulder, i.e., inboard of any nonzero lengths, in a nonredundant robot arm, the trick to solving for this joint is to realize that it must allow the plane of the arm to contain both the effective point and the wrist. The procedure

shouldertwist({x, y, z, theta, phi})

takes a list of world coordinates and returns the angle for the joint that determines yaw.

Examples of the construction of inverse kinematic procedures are given in Section 5.2.

Chapter 5

Applications of RobotScribe

In this chapter we describe some practical problems of robot design and show how our system can be applied to each. We work out each application first for a two-link planar elbow arm, for simplicity, and then show or sketch the application for more complicated arms.

5.1 Generating the Forward Kinematic Algorithm

Firmware for robot control must include a procedure for determining the position in world coordinates of the end-effector, given the state of the joint variables. To design such a procedure, one needs to formulate the forward kinematic equations as an algorithm.

The Kinematics Helper module of this project automatically generates the Forward Kinematic Equation in symbolic form, which is the backbone for the robot algorithm. Then the manipulative facilities of the computer-algebra environment help to put the pieces together.

The first step is to specify the robot. A two-link elbow manipulator in the horizontal plane with link lengths 15 and 12 is set up by the commands

```
initialize();
  revolute(z,15);
  revolute(z,12);
  generate();
```

The second step is to display the Robot Equation using showeq(), as shown in the first two frames of Figure 5-1.

The next step is to examine the results of showeq(). In this case we find we can simplify the expressions for x, y, and yaw by using the addition formulas for sine and cosine.

Finally, note that this configuration is ambiguous, as defined in Section 2.5. If V2's physical range includes both positive and negative values, the control algorithm will probably require specifying which one we have here. By using a template that RobotScribe provides, we can incorporate a conditional statement setting a variable to report the choice of V2.

The result, shown in the last frame of Figure 5-1, is the forward kinematic algorithm for this two-link arm.

5.2 Generating the Inverse Kinematic Algorithm

If the robot is to respond to input in world coordinates, firmware for robot control must include a routine for determining the joint variable values needed to put the end-effector in a position whose world coordinates are given.

First the designer must use his insight to analyze the arm into calculable subparts. In this case, it is easy to see that the arm consists of just an elbow assembly.

```
L30
showeq()
L31
[(15·cos(V1))+12·(cos(V1)·cos(V2)-sin(V1)·sin(V2))
 15·sin(V1)+12·(cos(V1)·sin(V2)+cos(V2)·sin(V1))
                        0
atan2(cos(V1)·sin(V2)+cos(V2)·sin(V1),cos(V1)·cos(V2)-sin(V1)·sin(V2))
                        0]
L32
[(15·cos(V1))+12·(cos(V1+V2))
 15·sin(V1)+12·(sin(V1+V2))
            0
atan2(sin(V1+V2),cos(V1+V2))
            0]
L33
[(15·cos(V1))+12·(cos(V1+V2))
 15·sin(V1)+12·(sin(V1+V2))
            0
          V1+V2
            0]
L59
<<
  x←(15·cos(V1))+12·(cos(V1+V2));
  y←15·sin(V1)+12·(sin(V1+V2));
  yaw←V1+V2;
  if numberp(V2)                ;
    then <<
           if V2>0              ;
             then q←pos
             else if V2=0
                    then q←zero
                    else q←neg
           nil
         >>

>>
```

Figure 5-1 Making the forward algorithm

The second step is use the template that RobotScribe provides for the inverse procedure. Editing the template results in the procedure shown in the first frame of Figure 5-2. Invoking this inverse procedure displays a list of expressions for the joint variables in terms of world coordinates, as shown in the third frame of Figure 5-2.

The next step is to examine these expressions for simplification. We can use the global renaming function in MathScribe to find automatically all occurrances of a selected subexpression. In this case we see that a subexpression in the expression for V1 can be replaced by V2. Thus we arrive at the formulas for V2 and V1 shown in the fifth and sixth frames of Figure 5-2.

Finally the control algorithm requires choosing between the positive and negative arccosines for V2, and also checking that each joint variable is within its physical range. These items are pulled in using templates that RobotScribe provides.

The result, shown in the last two frames of Figure 5-2, is the inverse kinematic algorithm for this two-link arm.

5.3 The Manipulator Jacobian

The manipulator Jacobian represents the infinitesimal relationship between the joint displacements and the end-effector location at the present position [Asada and Slotine, 1986]. The infinitesimal motion relationship is determined

```
L58
proc inverse (point);
  begin
    declare scalar x,y
    x←part(point,1);
    y←part(point,2);
    i←1;
    return elbow({x,y,i})
  end
```

L34

```
inverse({x,y})
```

L35

$$\{-\text{asin}\left(\frac{12\cdot\sin\left(\text{acos}\left(\frac{x^2+y^2-369}{360}\right)\right)}{\sqrt{x^2+y^2}}\right)+\text{atan2}(y,x),\ \text{acos}\left(\frac{x^2+y^2-369}{360}\right)\}$$

L44

$$V1\leftarrow-\text{asin}\left(\frac{12\cdot\sin\left(\text{acos}\left(\frac{x^2+y^2-369}{360}\right)\right)}{\sqrt{x^2+y^2}}\right)+\text{atan2}(y,x)$$

L45

$$V2\leftarrow\text{acos}\left(\frac{x^2+y^2-369}{360}\right)$$

L46

$$V1\leftarrow-\text{asin}\left(\frac{12\cdot\sin(V2)}{\sqrt{x^2+y^2}}\right)+\text{atan2}(y,x)$$

L55

```
<<
  V2←acos((x^2+y^2-369)/360);
  if q=neg        ;
    then V2←-V2
  if numberp(V2)                              ;
    then if or(V2<links(2,5),V2>links(2,6))
           then write(V2," out of range for Link ",2)

>>
```

L57

```
<<
  V1←-asin(12·sin(V2)/√(x^2+y^2))+atan2(y,x);
  if numberp(V1)
    then if or(V1<links(1,5),V1>links(1,6))
           then write(V1," out of range for Link ",1)
>>
```

Figure 5-2 Making the inverse algorithm

by differentiating the kinematic equations. The Jacobian has many uses in kinematic analysis.

One use is local linear approximation. The Jacobian is the vector analogue of the derivative of a real function of one real variable. Just as the derivative can be used to linearly approximate a real function in an interval, the arm Jacobian can be used to linearly approximate the Joint Space to World Coordinate Space transformation in a region .

Furthermore, at any point the Jacobian can be inverted, unless it is singular, to produce a local linear approximation of the more difficult World Coordinate Space to Joint Space transformation. This provides a shortcut when robot control requires calculating the joint variable changes required to make small corrections in the world coordinates of the end-effector. On robots where a closed-form inverse algorithm is not available, the Jacobian is a key element in numerical methods, such as the Newton-Raphson algorithm, for inverse kinematics.

Another use is finding singularities. Points where the Jacobian cannot be inverted, i.e., where it is a singular matrix, correspond to positions of the robot arm where the World Coordinates to Joint Space transformation is not defined. These are positions where degeneracy occurs (see Section 2.5), and are called singularities of the arm. In using a particular robot arm, it is important to locate the singularities of the arm in order to avoid their neighborhood in robot applications.

A matrix is singular when its determinant is zero, so by calculating the determinant of the Jacobian in symbolic form and solving to find its zeros, we find the singularities of the arm.

To use RobotScribe to derive the Jacobian of our two-link example, the first step is to use display the Robot Equation, as shown in Figure 5-3. Using MathScribe functions, a matrix (2 x 2 in this case) is set up, the expressions for x and y are copied in from the display, and the derivative operator is wrapped around each coefficient, with respect to V1 in the first column and V2 in the second.

MathScribe performs the differentiation, resulting in the manipulator Jacobian shown in the last frame of Figure 5-3.

To investigate singularities of the arm, we let REDUCE take the determinant of the Jacobian, as in Figure 5-4. In the case of this two-link arm, the determinant of the Jacobian is a multiple of sin(V2), so we see that the singularities of the arm will occur when the second joint variable is 0 or any multiple of pi, that is, when the arm is fully extended or when it is doubled back on itself. Similarly, the built-in matrix inversion in REDUCE give us the inverse Jacobian immediately, as shown in Figure 5-5.

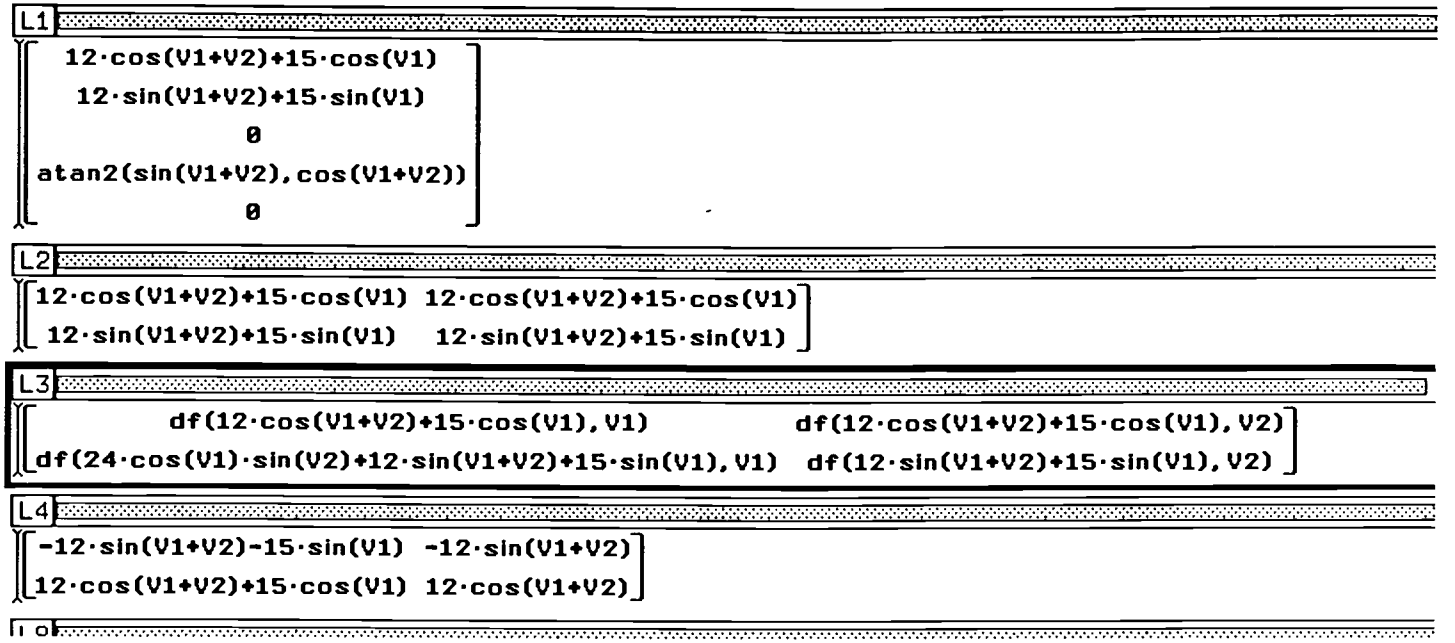


Figure 5-3 Deriving the Jacobian

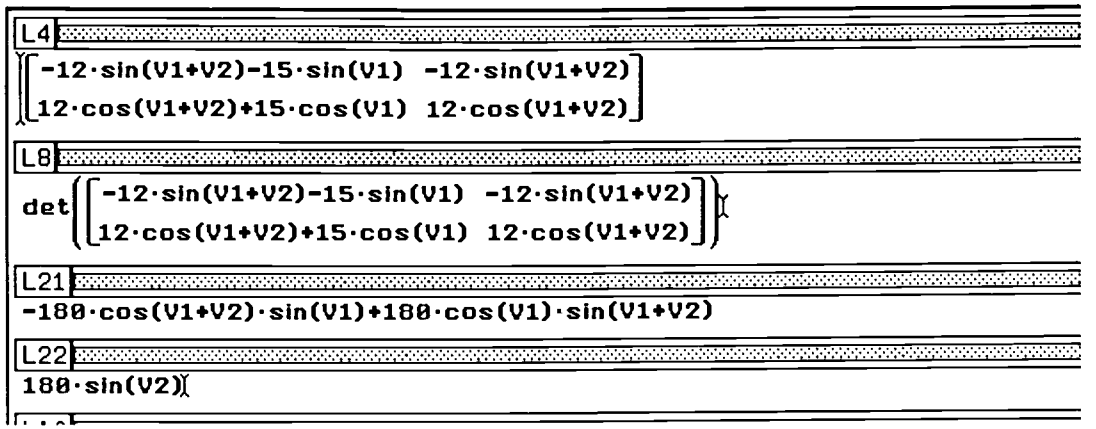


Figure 5-4 The Inverse Jacobian

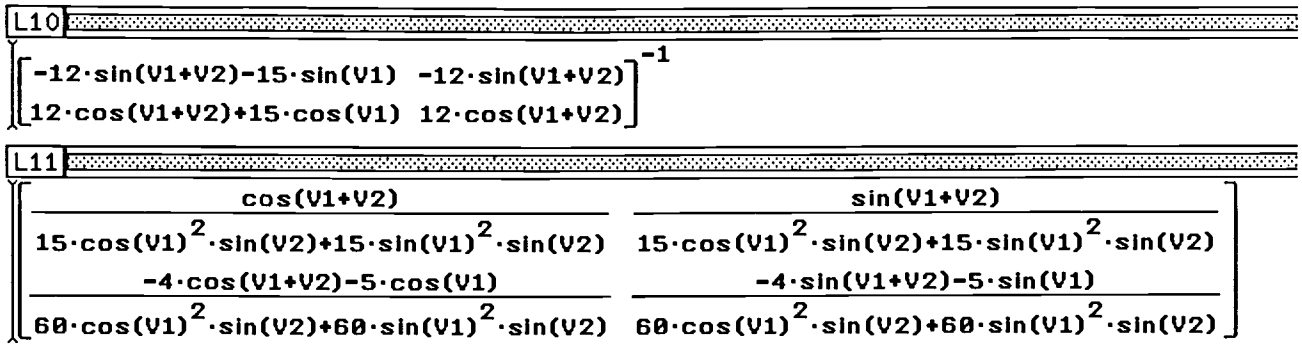


Figure 5-5 Using the Jacobian to Find Singularities

Chapter 6

Discussion

In this chapter, we summarize the contributions of this project and identify areas for future research.

6.1 Summary

This project is concerned with the application of computer algebra to the problems of robot arm kinematics.

In Chapter 2, we reviewed the basic kinematic concepts which are the framework for this project and we introduced orthogonal representation of a robot arm link, which is simpler than the usual Denavit-Hartenberg representation. In cases where full generality is needed, it can be regained with orthogonal representation by the addition of fixed links to the kinematic model of the robot arm. A noteworthy feature of this project is our descriptive formalism. Instead of the prevalent Denavit-Hartenberg generalized link nomenclature and homogeneous transformations, we use simpler and more intuitive, and yet sufficiently general, formulations. Chapter 3 is a brief discussion of the interactive, windows- oriented computer algebra system that is the platform for this project.

Chapter 4 presents our system of tools for robot kinematic analysis in symbolic as well as numerical form, and Chapter 5 demonstrates the use of our system in several practical problems on several robot arm configurations. The

most notable feature of our system is its symbolic capability. Other robot kinematic tools that we are aware of are limited to numerical output; our system allows the user to generate, display and algebraically manipulate the underlying mathematical equations that characterize the robot arm as a kinematic system. It provides tools for mathematical analysis, as well as numerical computation. For the design of robots, this is valuable because the symbolic form of the Forward Kinematic Equation and the Inverse Kinematic Equation are the basis for the algorithms that decide where to move the joints in order to achieve a desired position. For applications of robots, having equations in symbolic form fosters analysis of specific constraints.

6.2 Future Directions

We have demonstrated that computer-algebra tools can be integrated with more conventional robotic tools. At present, computer aids to mechanical design in general consist chiefly of graphic and numerical tools. One obvious direction to pursue is the integration of computer-algebra tools as part of a full-scale robot modeling environment, with advanced graphics and detailed robot simulation.

We have developed a set of tools to support kinematic analysis in an interactive, symbolic environment. Further work could identify and respond to more specific needs of those who design robots and robot applications.

Our attack on the Inverse Kinematic problem automates subsolutions for frequently-occurring subconfigurations. Further work could pursue the

automatic generation, for suitable robot arms, of Inverse Kinematic Equations by automating the analysis of an arm into its subconfigurations.

Our system generates the symbolic expressions that are the backbone of the Forward and Inverse Kinematic Algorithms for robot control; in a sense it "compiles" robot arm descriptions into mathematical algorithms. Further work could pursue the automatic generation of robot control code from arm descriptions.

Another direction is extension of our automatic generation technique for symbolic-form kinematic equations to the generation of dynamic equations in symbolic form. The dynamic equations, which describe the relation between the actuator joint torques and the resultant force and moment applied by the end-effector, form the basis for control of the force applied by a robot arm. Derived as are the kinematic equations in a link-by-link progression, the dynamic equations are considerably more complicated, with a high occurrance of repeated terms [Featherstone, 1987], suggesting the value of computer algebra tools.

We believe that progress in these directions will extend the contributions of this project, leading to further improvements in robot kinematic tools, and aid ease the development of robots and robot applications.

References

S.K.Abdali and G.W.Cherry,
"Visual Environments for Symbolic Computation",
Proc. European Conference on Computer Algebra, Linz, Austria,
April, 1985.

I.Asimov,
"Runaround",
Astounding Science Fiction, March, 1942.
(reprinted in I, Robot, 1950)

H.Asada and J.-J.E.Slotine,
Robot Analysis and Control, Wiley, 1986.

B.Buchberger, E.E.Collins, R.Loos and R.Albrecht, editors,
Computer Algebra Symbolic and Algebraic Computation,
Springer-Verlag, 1982.

K.Capek,
"R.U.R", translated from the Czech by P.Selver,
Washington Square Press, 1973.

J.T.Dallam,
"An Industrial Viewpoint Regarding University Robotics,
Education and Research",
Proc. Fifth National Conference on University Programs in Computer-Aided Engineering, Design, and Manufacturing, 1987.

J.Denavit and R.S.Hartenberg,
"A Kinematic Notation for lower pair mechanisms based on matrices",
ASME Journal of Applied Mechanics, 22:215-221, 1955.

A.C.Hearn, ed.,
REDUCE User's Manual, The Rand Corp., 1983.

Intelledex, Inc.,
RBASIC User's Manual, Intelledex, Inc., 1983.

R.Featherstone,
Robot Dynamics Algorithms, Kluwer Academic Publishers, 1987.

J.Meyer,
"An Emulation System for Programmable Sensory Robots",
IBM Journal of Research and Development, Vol. 25(6), 1981.

R.P.Paul,
Robot Manipulators: Mathematics, Programming and Control,
MIT Press, 1981.

M.S.Pickett, R.B.Tilove, and V.Shapiro,
"Roboteach, an Off-Line Programming System Based on GMSolid",
General Motors Research Lab. Research Publication, 1983.

D.L.Pieper,
"The Kinematics of Manipulators under Computer Control",
AIM 72, Stanford Artificial Intelligence Laboratory,
Stanford University, 1968.

T.Sata, F.Kimura, A Amano,
"Robot Simulation System as a Task Programming Tool",
11th International Symposium on Industrial Robots, 1981.

C.E.Smith,
personal communication.

C.J.Smith and N.M.Soiffer,
"MathScribe: A User Interface for Computer Algebra Systems",
Tektronix Technical Report, 1986.

R.N.Stauffer,
"Robot System Simulation",
Robotics Today, June, 1984.

Tektronix, Inc.,
MathScribe User's Manual, Tektronix, Inc., 1988.

K.J.Waldron,
"The Use of Computer-Aided Engineering Tools in the Design of Robotic Systems",
Proc. Fifth National Conference on University Programs in Computer-Aided Engineering, Design, and Manufacturing, 1987.

D.K.Weiland,
The Robotics Industry: Current Trends, GMI Engineering and Management Institute, 1985.