Automatic Kinematic Modelling of Robot Manipulators and Symbolic Generation of their Inverse Kinematics Solutions

(Extended Abstract)

Dan Halperin
Department of Computer Science
Tel-Aviv University
Tel-Aviv 69978, ISRAEL

Abstract

The increasing use of robot simulation systems has raised the need for a full kinematic treatment for a large number of different types of robots. Usually this is done either by supplying a general iterative inverse procedure, a time-consuming method not guaranteed to converge, or by carrying out a special analysis for each type. In this paper we present a software system whose input is an elementary geometric description of a robot. The system automatically builds the Denavit-Hartenberg model out of the input, while retaining information of how to interface between the built model and the elementary description. It then derives the equations of the inverse kinematics and subsequently solves them symbolically in closed form (if possible). The solution is accurate, efficient in on-line use, gives all possible solutions for every input frame of the end-effector and is thus suitable for use in robot simulation. Our software package was incorporated into the ROBCAD* simulation system where it has been successfully modelling and solving a myriad of kinematic structures of robots and other mechanisms.

1 Introduction

The advance of robotics and the increase in robot use have raised the need for computer simulation of robots, among the aims of which are the design of new robots, task planning of existing robots, performance evaluation and cycle time estimation. An important part of robot simulation is the treatment of robot kinematics. The various aspects of kinematic robot simulation are the concern of this report.

*ROBCAD is a trademark of Robcad Ltd., Herzlia

The basic kinematics problem is divided into two subproblems—the direct and inverse kinematics problems. Provided that the various geometric link parameters are given, the direct kinematics problem is to find the position and orientation of the end-effector, given the joint values of the robot. The inverse kinematics problem is to find the values of the joint variables for positioning the end-effector of the robot arm at a desired position and orientation.

We wish to provide a robot simulation user with a full kinematic simulation (direct and inverse) without requiring him or her to possess deep understanding of the kinematics problem. But then, two fundamental issues are raised: 1) How to build a kinematic model from geometric information such that it will allow for activation of the robot as well as for an efficient solution of the inverse kinematics problem. 2) How to solve the inverse kinematics problem automatically such that the solution obtained will be efficient in on-line use.

As was forecast by Denavit and Hartenberg [DH], their proposed model is most convenient for "electrical computation". Nevertheless, since their procedure is not intuitive and can be quite tedious and error-prone, there is still the need to hide the details of this inner representation of the kinematic model and enable the user to activate the robot merely through its geometric description. A more difficult problem is to supply a global inverse solution. One may wish to use a general iterative solution for this purpose (e.g., [UDH],[GBF],[GK]), which could be programmed to solve all types of robots. Such a solution, however, is time-consuming, there is no guarantee of convergence and, in most cases, not all the possible solutions for a given input are obtained. Another approach, using a closed-form solution (e.g., [Pi],[Pa],[PSM],[Le]), which does not suffer from these shortcomings, requires a special analysis for each type of robot.

In what follows, the two fundamental issues previously mentioned are discussed and two programs written to solve the related problems are described. The first program, the Kinematic Model Compiler (the KMC), transforms an elementary geometric robot description into the Denavit-Hartenberg (DH, for short) model of that robot, while retaining an interface between the DH-model and the original description. (Thus, the user of our software does not need to understand the DH model.) The KMC is completely general and together with a simple extension to the basic DH-model it can model every "tree-like" mechanism and is not limited to kinematic chains. The second program, the Inverse Solution Generator (the ISG), is a high-speed special-purpose symbolic computation system deriving its inverse kinematics equations from the output of the KMC and solves them in closed form (if possible). Our symbolic inverse solver exploits the special structure and characteristics of the inverse kinematics equations and uses efficient algorithmic techniques to obtain solutions within few seconds. See Figure 1 for a schematic description of our software system.

In a work that has been recently published ([HMC]), a system similar to ours is presented. Our system, however, has the advantage that it also deduces the DH-parameters automatically. Moreover, the implementation of [HMC] is significantly slower than ours, e.g., the software of [HMC] solves the direct and inverse kinematics of the Stanford manipulator in 862 seconds on a Data General

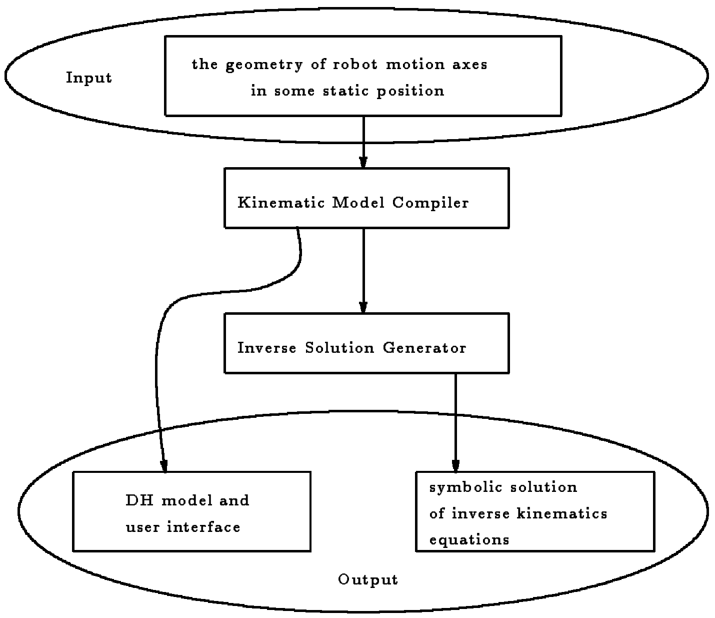
Input
the geometry of robot motion axes in some static position
Kinematic Model Compiler
Inverse Solution Generator
DH model and user interface
symbolic solution of inverse kinematics equations
Output

Figure 1: The software process for one robot

MV10000, whereas our software solves the same robot in 7 seconds on an entry-level workstation SUN 3/50. Lastly, we remark that our work was first reported in 1986 ([Ha]) and so it was probably the first software system to solve the inverse kinematics problem symbolically.

In Section 2 we give a description of the Kinematic Model Compiler and point out two uses of its results. Section 3 is devoted to the Inverse Solution Generator with an emphasis on the underlying algorithmic techniques. A short summary and some concluding remarks are given in Section 4.

This abstract is far too brief to give a complete view of our work. A full description, including many details, can be found in [Ha].

2 Kinematic Model Compiler

The Denavit-Hartenberg model ([DH]) gives a solution to the direct kinematics problem and provides a convenient formulation of the inverse kinematics equations. We omit a review of the DH parameters and refer the reader to any textbook on robotics (e.g., [Pa],[Cr]).

The Kinematic Model Compiler (KMC, for short) is a software tool that turns an elementary geometric description of a robot into the DH model of the robot. This internal DH model allows for an efficient activation of the robot (in a simulation system, for example) and prepares the input for the second automatic part of our software system—the symbolic inverse kinematics solver, described in Section 3. The input to the KMC is the geometry of the motion axes of a robot, or joints, in some static position. The main advantage of the KMC is that its input can be easily obtained from most existing CAD systems (or given by a simple textual input) and the user of our software does not need to understand the kinematic model and the DH parameters.

2.1 The Algorithm

The necessary data for building a kinematic model and for performing kinematic simulation of a robot are: For every link: a coordinate system attached to the link. (All the links can be described in one "universal" system and in such a case no link information is needed.) For every joint: (i) its type: either revolute or prismatic; (ii) the line along which it is defined; and (iii) the two links that it connects. And this is also the input to the KMC. The coordinate frame of a link and the straight line of a joint are both given in some static position of the robot. The input is given as a program in a simple structured language.

In a preparatory stage of the KMC, lexical, syntactical and preliminary-semantical analyses of the input program are executed. As a result, a tree-like data structure is constructed (see below for more details) and right afterwards three passes on this data structure are performed for (i) building the new coordinate frames, i.e., the DH frames; (ii) asserting the DH parameters $(a_i,\alpha_i,d_i,\theta_i)$; and (iii) computing the interface transformations between the user model and the DH model.

We skip the details of the three passes and turn to describe the output of the KMC. It consists of two sets of transformations and a special tree data structure. The sets of transformations are:

1. The DH matrices (or A-matrices), $A_i, i = 1, \ldots, n$
2. The interface transformations, $HTL_i, i = 0, \ldots, n$, where HTL_i is a matrix that transforms the original (user) coordinate frame of link i to the DH coordinate frame of link i.

The data structure produced by the KMC is a tree-like structure where a node is allocated for the data of a link and for the joint which connects this link and its father link. There is an edge between two nodes if their links are connected by some joint. It is not a standard tree data structure because our extension to the basic model to make it suitable for tree-like mechanisms (which, for lack of space, we do not describe here) results in duplication of nodes and direct connections between siblings in it.

The first use of the KMC output is for activating a robot. Given the joint values of the robot (the input of the direct kinematics) we substitute for these values in the A-matrices, now all the matrices are known and we have to compute the suitable transformation for each link. For link i the computation is $BTW \cdot HTL_0^{-1} \cdot A_1 \cdot A_2 \ldots A_{i-1} \cdot A_i \cdot HTL_i$, where BTW is a transformation which enables us to move the robot in its world (for a robot on a rail, for example). This computation may be carried out using the same scan procedure with which the model was built. (In the full version of this paper we consider some efficiency issues regarding this computation.) The second use of the KMC output is the definition of the DH parameter table for use by the inverse solver.

2.2 Implementation Details

The preparatory stage of the KMC (the lexical and syntactical analysis) is performed by a *Recursive Descent Compiler* (see [AU]). The grammar of the input language appears in [Ha].

The geometric computations are mainly performed by two packages. One package deals with numerical manipulation of homogeneous transformations: transformation multiplication, homogeneous matrix inversion (a simple operation relative to general matrix inversion), construction of a coordinate frame from given points, and more. The other geometric package handles lines in space represented in *Plücker coordinates*, which is a convenient representation for computing the common-normal of two given lines, the twist angle between the lines and the distance between them. See [So] for a basic discussion of Plücker coordinates and [BR],[Sh] for their use in kinematics.

The three passes on the tree data structure are performed by the same traversal procedure that scans the tree *depth first*. In each pass the procedure is input the function which should be operated at each node, one function per pass.

The KMC is written in the "C" programming language. Its source code is 5,000 lines long. The average running time of the KMC on a six degrees-of-freedom robot is 2 seconds on a SUN 3/50 workstation.

3 Inverse Solution Generator

The Inverse Solution Generator is an "expert-system" that follows the closed-form solution approach. Its set of rules for solving the kinematics equations, called here "solving patterns", were collected from the kinematics literature and were put into the system in the most general form still feasible. Few other rules were added as a by product of the system development process. We do not list out the solving patterns that the program uses (a full list of the rules is given in [Ha]), but rather emphasize the algorithmic aspects of the ISG.

3.1 The Algorithm

The following is a sketch of the algorithm:

- *Phase 1.* Transforming the DH-parameter table into symbolic sets of equations of the form

$$\begin{aligned} A_1 A_2 \ldots A_n &= T \\ A_2 \ldots A_n &= A_1^{-1} T \\ &\vdots \\ A_n &= A_{n-1}^{-1} A_{n-2}^{-1} \ldots A_1^{-1} T, \end{aligned}$$

  where T describes the desired location of the end-effecot. When these sets are insufficient, the "reverse" sets are produced (following an idea in [PSM]).

- *Phase 2.* Producing the solution sequentially. The solution procedure is iterative (not in the numerical sense, of course). In each iteration all instances of the already solved joint variables are considered constants. The equations are scanned using a set of rules for searching patterns that will lead to a solution. Once such a pattern is found, it becomes a solution-suggestion for the current step. A suggestion gets a score based on the validity of the different solutions that stem from it and on its numerical stability. The moment a suggestion with the highest possible absolute score is found, the search stops and this suggestion is chosen. Otherwise, after scanning all the equations, the suggestion with the best score is accepted. In either case, the selected suggestion is turned into the solution of the relevant variable. We mark this variable as solved and proceed to the next iteration.

  This phase ends when all the variables are solved or when there are no suggestions in the last iteration.

- *Phase 3.* The one-step solutions are printed in the order of their production, yielding the sequential procedure of the overall solution.

There are several aspects of the algorithm that we do not discuss in detail in this short report. We mention that the algorithm efficiently copes with "trivial" equations such as $(p_z - d_1)\cos\theta_4 - (p_y + d_2)\sin\theta_4 = 0$ when the two variables d_1, d_2 have already been solved by the equations $d_1 = p_z$ and $d_2 = -p_y$. The algorithm is also capable of producing partial solutions by "locking" some of the joint variables and so it can generate solutions for a rapidly converging iterative method suggested by Lumelsky ([Lu]) when a closed-form solution cannot be obtained.

3.2 Implementation Details

Since our software system was designed for on-line use it was important that it should work fast. It was not hard to design the kinematic modeller so that it would work fast but it was a difficult task to obtain a similar speed for the inverse solver. Evidently, using an existing general-purpose symbolic computation software would not have achieved the required speed and so we turned to design a special-purpose symbolic software that takes advantage of the special characteristics of the problem at hand.

A major part of the ISG is a LISP emulator in the "C" language. Our LISP package uses a standard binary tree structure to represent lists in LISP, where an internal node is of type "list", and its leaves are the LISP atoms. Our package implements all basic LISP functions (car, cdr, etc.), and, on top of them, a battery of functions to perform the symbolic algebra, namely, addition, multiplication, matrix multiplication, homogeneous matrix inversion, simplification and term cancellation.

An equation is a combination of two list-trees, one for the expression of each side. A matrix equation is a set of twelve non-trivial equations. Once the equations were built using the list structure, we wish to operate on them. The operations are: searching for solving patterns, searching for equations with a small number of variables and comparing expressions and subexpressions. The algorithmic tool we now turn to describe made these operations simple and efficient.

For every node in the list-binary-tree we attach a computer-word where a bit is turned on (i.e., its value is set to 1) for every variable that appears in the subtree of that node. We call this computer-word a *variable mask*. Since the expression trees are built bottom-up it is easy to update the masks. The variable masks, together with a special mask where there is a bit turned on for every variable that has not yet been solved, are used for a quick filtering out of candidates for solving patterns. Moreover, they serve as a means for canonical representation of expressions which further contributes to the efficiency of the search for solving patterns. We refer the reader to the full version of the paper for more details on the variable masks.

The resulting program can symbolically solve the inverse kinematics equations of a six-degrees-of-freedom robot in few seconds (approximately 8 seconds on a SUN 3/50 workstation). The ISG is written in the "C" programming language and its source code is 7,000 lines long.

4 Conclusion

In this paper we have presented a software system that automatically models mechanisms and then symbolically solves them in closed form, whenever possible. The first part of the system, the Kinematic Model Compiler, cam model every tree-like mechanism while completely freeing the user of the software from understanding the kinematics issues. The second part, the inverse solution generator, has been successful in solving the inverse kinematics for a large number of robots; it has solved all the robots for which a closed-form solution is well-known and many more.

As mentioned before, our software was incorporated into the ROBCAD simulation system ([Ho]). There it was elaborated in several directions; to mention a few: solution of redundant robots; supplying a collection of solutions for robots with less than six degrees of freedom by inputting partial end-effector's frames; on-line activation of the symbolic solutions of the ISG. Another possible enhancement of the ISG is a solver of the inverse kinematics of velocity using an additional set of rules, based on a technique described in [Pa].

We believe that beyond its already proven "applied" value, our software can serve as a convenient research tool for the study of kinematics problems.

Acknowledgements

I am grateful to Micha Sharir for his advice during the software development and for his comments on the original report ([Ha]). Sergio Lanzman and Ofer Lev had studied the software and suggested several improvements. Special thanks to Sally Jacoby for her assistance in preparation of this report.

References

[AU] A.V. Aho, J.D. Ulmann, *Principles of Compiler Design*, Addison-Wesley, 1977.

[BR] O. Bottema, B. Roth., *Theoretical Kinematics*, North Holland Publishing Company, 1979.

[Cr] J.J. Craig., *Introduction to Robotics*, Addison Wesley, 1986.

[DH] J. Denavit, R.S. Hartenberg, A Kinematic Notation for Lower Pair Mechanisms Based on Matrices, *ASME J. of Applied Mechanics*, June 1955, pp. 215-221.

[GBF] A. A. Goldenberg, B. Benhabib, R. G. Fenton, A Complete Generalized Solution to the Inverse Kinematics of Robots, *IEEE J. of Robotics and Automation*, Vol. RA-1, No. 1, March 1985, pp. 14-20.

[GK] K. C. Gupta, K. Kazerounian, Improved Numerical Solutions of Inverse Kinematics of Robots, *Proc. IEEE International Conference on Robotics and Automation*, 1985, pp. 743-748.

[Ha] D. Halperin, *Kinematic Modelling of Robot Manipulators and Automatic Generation of their Inverse Kinematics Solutions*, M.Sc. Thesis, Tel-aviv University, August 1986. *In Hebrew* (forthcoming in English).

[Ho] J. Holland, *ROBCAD Reference Manual*, Robcad Ltd., Herzlia, 1986.

[HMC] L.G. Herrera-Bendezu, E. Mu and J.T. Cain, Symbolic Computation of Robot Manipulator Kinematics, *Proc. IEEE International Conference on Robotics and Automation*, 1988, pp. 993-998.

[Le] C.S.G. Lee, Robot Arm Kinematics, *IEEE Tutorial on Robotics*, 1983, pp. 47-65.

[Lu] V.J. Lumelsky, Iterative Coordinate Transformation Procedure for One Class of Robots, *IEEE Trans. on Systems Man and Cybernetics*, Vol. SMC-14, No. 3, May 1984, pp. 500-505.

[Pa] R.P. Paul, *Robot Manipulators, Mathematics, Programming and Control*, The MIT Press, 1981.

[Pi] D.L. Pieper, *The Kinematics of Manipulators Under Computer Control*, Ph.D. Thesis, Stanford University, 1969.

[PSM] R.P. Paul, B. Shimano, G.E. Mayer, Kinematic Control Equations for Simple Manipulators, *IEEE Trans. on Systems Man and Cybernetics*, Vol. SMC-11, No. 6, June 1981, pp. 456-460.

[Sh] B.E. Shimano, *The Kinematic Design and Force Control of Computer-Controlled Manipulators*, Ph.D. Thesis, Stanford University, 1978.

[So] D.M.Y. Sommerville, *Analytical Geometry of Three Dimensions*, Cambridge University Press, 1959.

[UDH] J.J. Uicker Jr., J. Denavit, R.S. Hartenberg, An Iterative Method for the Displacement Analysis of Spatial Mechanisms, *Trans. of the ASME, J. of Applied Mechanics*, Vol. 31, June 1964, pp. 309-314.