

# Automated Kinematic Generator for Surgical Robotic Systems

David L. JUNG, Warren E. DIXON, François G. PIN  
*Engineering Science and Technology Division*  
*Oak Ridge National Laboratory*  
*Oak Ridge, TN, USA*

\*Oak Ridge National Laboratory is managed and operated by UT-Battelle, LLC for the U.S.  
Department of Energy under contract DE-AC05-00OR22725

**Abstract.** Unlike traditional assembly line robotic systems that have a fixed kinematic structure associated with a single tool for a structured task, next-generation robotic surgical assist systems will be required to use an array of end-effector tools. Once a robot is connected with a tool, the kinematic equations of motion are altered. Given the need to accommodate evolving surgical challenges and to alleviate the restrictions imposed by the confined minimally invasive environment, new surgical tools may resemble small flexible snakes rather than rigid, cable driven instruments. Connecting to these developing articulated tools will significantly alter the overall kinematic structure of a robotic system. In this paper we present a technique for real-time automated generation and evaluation of manipulator kinematic equations that exhibits the combined advantages of existing methods – speed and flexibility to kinematic change – without their disadvantages.

## 1. Introduction

Robotic surgical assist systems commonly utilize a manipulator with multiple articulated joints. The next generation of these tools need to address evolving surgical challenges such as the restrictions imposed by the confines of a minimally invasive surgical environment, remote tele-surgery and supervised semi-autonomous systems. Consequently, new systems may involve many more joints – perhaps more resembling small flexible snakes rather than rigid cable-driven instruments – and an array of end-effector tools, which may themselves be articulated. The addition and removal of end-effector tools represents a significant change to the kinematic structure of the manipulator.

The control of articulated manipulators of this type requires continuous real-time evaluation of a forward kinematic transformation function  $h(q) \in R^n$ , where  $q \in R^n$  represents the vector of joint parameters values, and a matrix of its partial differentials  $J(q) \in R^{m \times n}$  – commonly called the system Jacobian. This is typically achieved by the control system designer writing out analytical expressions for  $h$  based on the manipulator kinematics and then differentiating to obtain  $J(q)$ . The complexity of the symbolic Jacobian expression increases rapidly in the number of joints. Hence, there are two common approaches implementers use to obtain a symbolic expression for  $J(q)$  that can be evaluated on-line. The first involves using a mathematical software package to symbolically differentiate  $h$  for a given manipulator off-line (e.g. Matlab<sup>®</sup> or Mathematica<sup>®</sup>). The resulting expressions, often much simplified due to cancellation and terms being multiplied by 0, is hand coded into the controller program code. The advantage of this method is speed, but at the cost of

fixing the manipulator kinematics. The second method involves hand coding program code that evaluates the general form of the Jacobian expressions, with both the kinematic parameters and the joint configuration parameters as variables. This has the advantage of flexibility to changes in the manipulator kinematics and relieves the burden of off-line derivation of the Jacobian expressions (which can easily run into many pages of equations for a manipulator with a modest number of links). However, it has the great disadvantage of a significantly increased operation count – as in typical cases, many numerical computations are performed only to be multiplied by 0 as the computation progresses. In this paper, we present a technique that retains the benefits of both approaches – accommodation of kinematic changes, avoidance of unnecessary on-line computations and automatic generation of Jacobian expressions.

## 2. Method

The system we have developed achieves both speed and flexibility by representing the forward kinematics expressions symbolically and analytically differentiating them on-line. Differentiation is computationally expensive, however it only needs to occur when the robot kinematic structure changes – such as when a tool is connected or disconnected from the end-effector. For example, a 6-dof manipulator Jacobian takes approximately 0.04 seconds to generate on an Intel Pentium IV equipped PC. During the normal control loop the symbolic expressions for the Jacobian only need to be evaluated for the given joint parameter values.

A tree structure represents the expression for each element of the forward kinematic vector and manipulator Jacobian matrix. The nodes of the tree correspond to operators and the leaves correspond to constants or variables. For example, the expression  $0.6\cos(\theta_4)$  can be represented as the tree shown on the left of the figure.

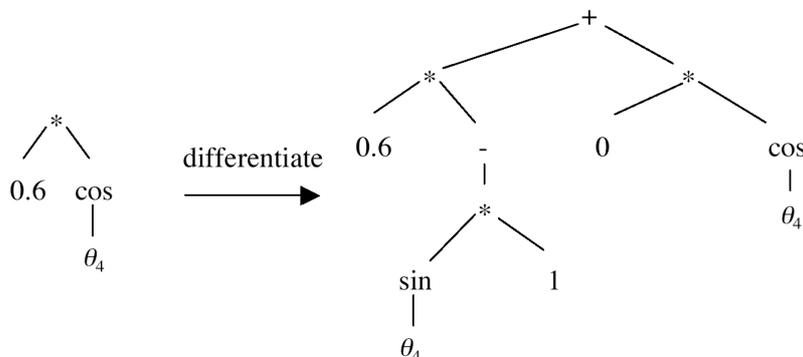


Figure 1. Simple expression tree and its differential tree

The software is implemented in the object-oriented C++ language. A class hierarchy exists that corresponds to the expression trees – with derived classes for concrete operators and leaf types. Each node overrides a polymorphic method that knows how to differentiate its specific type. For example, all the binary operator nodes achieve differentiation by straight forward application of the chain rule. The representation and manipulation of trees is wrapped in an `Expression` class that behaves like a value type – it has all the usual operators overloaded, can be passed efficiently by value and generally used like the built-in types such as `double`. This allows the re-use of the system vector and matrix classes with symbolic values as with regular value types. The right side of the figure shows the differential tree of the example expression.

The forward kinematics expression vector is typically constructed via multiplication of 4x4 transformation matrices for each joint. Consequently, the expression trees contain many nodes, with increasing complexity for the Jacobian. To reduce the differentiation and

subsequent evaluation of nodes that don't contribute to the value, expression simplification methods were implemented. These prune the expression trees in various ways corresponding to simplifications such as; elimination of multiplication by 1 or addition/subtraction of 0, removal of double negations, and constant subexpression evaluation. In our example differential tree, the simplified tree would correspond to the expression  $-0.6\sin(\theta_4)$  which is more efficient than evaluating  $0.6-(\sin(\theta_4)-1)+0\cos(\theta_4)$ .

The Jacobian matrix has one row per degree-of-freedom of the end-effector and one column per joint. Once the symbolic Jacobian has been constructed via differentiation as described, then the main control loop only needs to evaluate each element given particular values for the joint parameters in order to compute the inverse kinematics. The evaluation is performed by traversing the tree in a depth-first manner while passing the values of the variable parameters. Each node knows how to compute its value by first evaluating its children, if any. For example, a variable parameter leaf node evaluates to the current passed value of the corresponding joint variable, while an addition node evaluates to the sum of the values of its two children. Although, after simplification, the number of operations needed for evaluation of the whole matrix is comparable to the direct hand-coded format, there is a fixed overhead associated with tree traversal. Specifically, each operation and leaf evaluation incurs the costs of a polymorphic (virtual) method lookup and call. This adds a constant multiple of the number of operations to the evaluation cost, however as the number of joints increases it becomes insignificant in comparison to the number of operations saved by doing symbolic differentiation and simplification rather than exhaustive numeric evaluation.

### 3. Summary

By automatically constructing symbolic expressions for the forward kinematics of a manipulator and differentiating and simplifying to obtain the Jacobian on-line, the technique presented can easily adapt to changes in kinematic structure while maintaining computational costs akin to the direct numerical evaluation approach used for fixed kinematic systems. This capability will be instrumental in the successful engineering of control systems for surgical manipulator systems with many joints and a potentially infinite repertoire of articulated tools.

The system developed is currently in use as part of an inverse kinematics solver (IKOR[1]) used for redundant manipulator simulation and path planning – via our OpenSim[2] robot simulation framework which is capable of real-time and off-line 3D static and dynamic multiple robot simulations. Further details on the implementation of the symbolic Jacobian expression representation, construction, differentiation and simplification are available in [3].

### Acknowledgments

This research was supported in part by the U.S. DOE Office of Biological and Environmental Research (OBER) Environmental Management Sciences Program (EMSP) project ID No. 82794 at ORNL.

### References

- [1] François G. Pin and Faithlyn A. Tulloch, “**Resolving Kinematic Redundancy with Constraints Using the FSP (Full Space Parameterization) Approach**”, *Proceedings IEEE international Conference on Robotics and Automation (ICRA)*, 1996
- [2] **OpenSim** robot control and simulation framework. Source code and documentation can be obtained from the web-site: <http://opensimulator.sourceforge.net>.
- [3] K. Fischer, D. Jung, A. Cordero, W. E. Dixon, F. Pin, “**JFKengine: A Jacobian and Forward Kinematics Generator**”, *Oak Ridge National Laboratory Report ORNL/TM-2002/230*, 2002.