31st Midwest Symposium on Circuits and Systems

August 9–12, 1988
Marriott's Pavilion Hotel
St. Louis, Missouri

R. Eugene Stuffle
Linda R. Laub
Editors

UNIVERSITY OF MISSOURI-ROLLA

IEEE

AN EFFICIENT ALGORITHM FOR ROBOT MANIPULATORS DYNAMIC COMPUTATION

J. T. Machado[1) 2)], J. S. Matos[1) 2)], J. M. Carvalho[1) 2)] and A. C. Costa[2)]

1) School of Engineering of the University of Porto
Dept. of Electrical and Computer Engineering
4099 Porto Codex, Portugal

2) INESC Norte
Largo Mompilher 22
4000 Porto, Portugal

ABSTRACT

Despite recent advances in robot manipulator inverse dynamic computing methods, the development of industrial implementations of real-time controllers is still highly difficult, and impractical, due to the large number of arithmetic and transcendental floating point operations involved. The paper presents a method based on the use of a combination of boolean and ordinary algebra that leads to a considerable reduction of the on-line computing time. Terms of the link torque equations are calculated by boolean algebra, using Binary Decision Diagrams, and are then combined by arithmetic summations. The result is an efficient computational technique for real-time robot manipulator nonlinear control which is well suited to today's digital technology. The method, which is not restricted to the particular application described in the paper, is well adapted to implementation on a parallel architecture with performance improvement proportional to the number of processors, and without scheduling problems.

1. INTRODUCTION

The task of computing the required joint torques for a given set of positions, velocities and accelerations (i. e. the so called robot manipulator inverse dynamics problem), has been an area of active research for the last two decades. A topic of great concern, on which most research effort has been invested, is the development of techniques to reduce the complexity, and increase the efficiency, of the computations involved, in order to allow for real time execution of the resulting robot manipulator control programs, in industrial controllers.

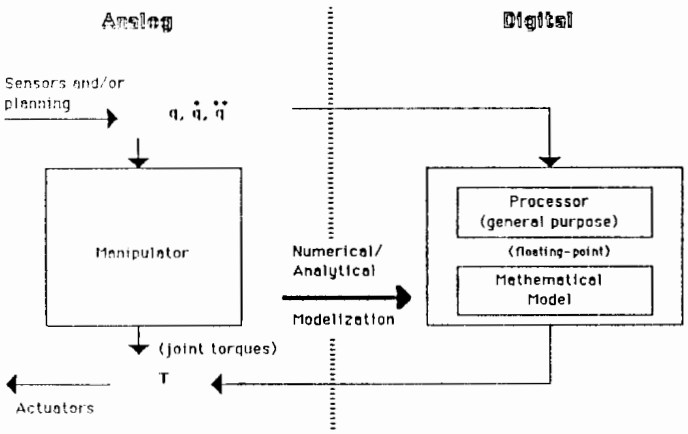
Systematic, Lagrangian-based methods were proposed since around 1960, and an optimal solution, for this particular formulation, was obtained by Hollerbach in 1980 [1]. By the same time Luh, Walker and Paul [2] presented an alternative method (Newton-Euler), derived from classical dynamics, which was shown by Silver [3] to be equivalent to the former. Although more efficient, it shared with Hollerbach's algorithm the need to evaluate a computationally very expensive set of recursive numerical expressions.

More recently (1986), the automatic generation of the dynamic symbolic formulae, using LISP-based algebra systems [4-5], has become possible, leading to a considerable improvement (up to one order of magnitude) in computational efficiency and performance. Symbolic methods allow a better insight into the manipulator dynamics [6]. They suggest new strategies for the design of the manipulator itself and customization of these expressions for specific robots [7-8], has allowed additional improvements in performance. More importantly though, the resulting non-recursive analytical expressions lead more easily to parallelism, a fact whose consequences are now being actively explored.

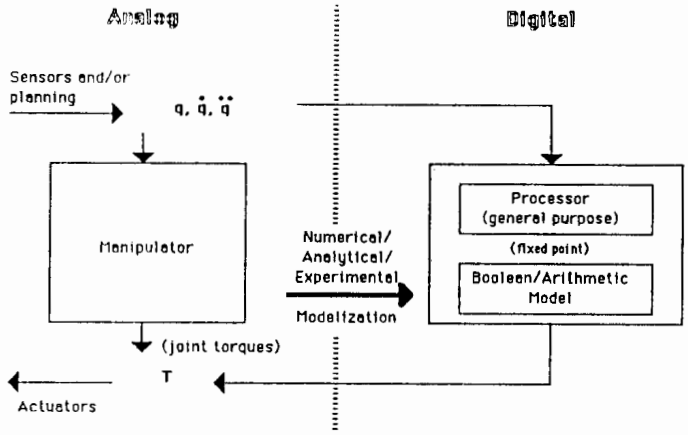
Robot manipulator control programs result from the coding of these complex analytical expressions in a high level language, and their translation, by means of a general purpose compiler, into an object code that runs on the target controller (usually a high performance CPU). The large number of floating point operations and transcendental function evaluations translates into a much larger number of processor instructions and machine cycles. Furthermore, the floating point operations are performed with precision (and word length) much larger than the one associated with the accuracy of typical manipulator hardware (sensors, A/D and D/A converters, etc.); clearly, calculation time would decrease if computations could be performed with precision matched to the accuracy of the manipulator hardware.

The algorithm proposed in this paper addresses both points. In a way, the basic idea is similar to the one behind transform calculus (e.g. Laplace, Z-transform): instead of performing computations in the time and analog domain, where the problem is most easily formulated, we choose to perform simpler evaluations in an algebra whose basic operations are more adequate to the current processors instruction sets. Furthermore we can use the precision that is appropriate for the problem (not more, not less), given the amplitude ranges expected or known for the variables, and the available accuracy of the robot manipulator hardware.

Boolean algebra seems a natural choice for this purpose. If both input and output variables are properly quantified and converted to binary, by means of a suitable



a) Conventional computational model



b) Proposed hybrid computational model.

Fig. 1. Computational model. a) Conventional; b) Proposed.

code, we can look at the resulting (potentially gigantic) table as a complete description of the on-line computations involved. Input variables on this truth-table are the bits of the quantified binary coded position, velocity and acceleration vectors. Output variables are the bits of the binary coded actuator torques. On this truth-table standard boolean minimization techniques could be performed, leading to simplified expressions for each bit of the required torque. A closer look reveals however, that this approach would be impractical, due to the size of the problem. (If the intrinsic input variables and output torque were coded with 8 bits each, this method would require, for a robot with 6 degrees of freedom, 144 input bits and 48 output bits). The strategy we followed [9] considers each link torque term separately, and leads to a hybrid algorithm that calculates each term by Boolean algebra, and combines them by means of the required arithmetic sums, to obtain each actuator torque. Details of the algorithm are given in section 2, and its application to a 2R robot manipulator is shown in [10]. Implementation issues, taking advantage of parallelism and specialized hardware are discussed in the last section.

2. THE ALGORITHM

General formulation

The dynamic equations for a manipulator with n degrees of freedom (d.o.f.) can be written in the form:

$$T=J(q).\ddot{q} + C(q,\dot{q}) + G(q) \qquad (1)$$

where

$J(q)$ is the nxn inertial matrix,

$C(q,\dot{q})$ is n-dimensional vector of Coriolis and centripetal torques,

$G(q)$ is the n-dimensional vector of gravitational torques

q, $\dot{q}$, and $\ddot{q}$ are the n-dimensional vectors of link positions, velocities and accelerations

The algorithm uses boolean algebra to compute each (product) term, and binary arithmetic to add them for computing the link torques. A fully boolean-based computation for a robot with n d.o.f. would require 3n input word variables (position, velocity and acceleration), each one quantified in an appropriate number of bits. By splitting each link torque in its terms, the size of the problem is considerably reduced. The truth table for computing the inertial terms will require at most n+1 input word variables; computation of the Coriolis, centripetal and gravitational terms will require, in the worst case, tables with n+2, n+1, and n input word variables, respectively.

The number of bits required to quantize and code the input variables and the link torque terms, is chosen in a way that takes in consideration the amplitude range, the available precision, and the required output resolution. Appropriate normalization (offset plus scaling), allows for the sums to be performed in fixed point arithmetic. Extra accuracy bits are used in order to compensate for the degradation in overall precision brought about by the summation operations. If p is the number of terms to be added, and an overall output resolution of r bits is desired, a total of w=r+m bits is used for each term, where the number of extra bits is given by

$$m = \text{int}[\log_2(p+1)] \qquad (2)$$

The "off-line" stage of the algorithm we have just described is dominated by the generation of the tables and their minimization. Obtaining the tables for the boolean evaluation of the torque terms, is a process that benefits from a priori knowledge of the mechanical and digital hardware, for the selection of the appropriate ranges, amplitudes, precisions and resolutions. This process can make use of the analytical expressions known for a particular manipulator. It is worth noting though, that the tables can be generated from purely experimental data, without the need to make use of any mathematical model. This is of particular relevance since situations exist where mathematical models are inaccurate or difficult to derive.

Table minimization may be performed using standard boolean minimization techniques like the tabular method of Quine-McCluskey [11]. The resulting expressions will then be compiled, or assembled, into the controller object code program, for on-line execution. An alternative, more efficient method, based on Binary Decision Diagrams, is described next.

Binary Decision Diagrams

Standard boolean minimization techniques typically lead to two-level, sum-of-products (or product-of-sums) expressions that, in the context of this work, are responsible for two types of limitations:

1) General purpose processors are not optimized for efficient realization of boolean operations. Processors with 8, 16 or 32-bit data buses, with large instruction sets (unused in this algorithm), and requiring several clock cycles for AND, OR and arithmetic sums, are largerly wasted in applications requiring the evaluation of bit oriented boolean formulae.

2) Boolean formulae are not the best way to evaluate boolean expressions. In fact, it was shown in [12] that, for n-variable functions, up to $O(2^n/\log n)$ operations may be required, while a method based on decision diagrams will never require more than n variable evaluations.

In order to avoid these shortfalls, Binary Decision Diagrams (BDDs) [13-14] are used in the (off-line) boolean minimization, and in the (on-line) function evaluation stages of the algorithm. BDDs are directed acyclic graphs which achieve extremely efficient representations of complex boolean functions. One node in the graph is a decision element controlled by one input variable; as such, nodes in the diagram can be directly translated into *if...then...else...* statements. Evaluating a diagram, i.e. finding the value of a function for a given set of inputs, requires in the worst case a number of decision tests equal to the number of input variables. For BDD synthesis, our method uses the simplification strategy and the algorithms given in [15]. By using this technique, we achieve automated synthesis, compact storage for the simplified truth-tables and optimal function evaluation performance.

Results

Figure 2 shows a log-log "hi-lo" chart comparing the histograms of the times required to compute several terms using the conventional arithmetic method (vertical axis) and the proposed algorithm (horizontal axis). By a and b we represent the terms $m_2 r_1 r_2 S_2 \dot{q}_2^2$ and $g(m_1+m_2)r_1 C_1$ of T_1, and by c and d the terms $(m_2 r_2^2 + J_2)\ddot{q}_2$ and $m_2 r_1 r_2 S_2 \dot{q}_1^2$ of T_2, respectively. The codes were written in Turbo Pascal V4.0, and examples were run on 8086, 8 MHz machine, under MSDOS V3.2. The simple term c imposes a similar load to both methods; the more complex terms a, b and d show a remarkable improvement when using the new algorithm. For more complex terms, like those appearing in manipulators with more d.o.f., the speed-up factor will be correspondingly higher.

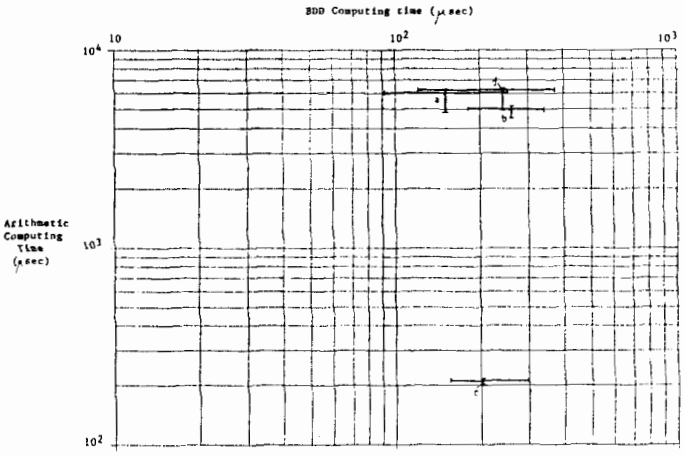


Fig. 2. Arithmetic vs. BDD computing time for several terms (see text). The "hi-lo"bars are centered on the average calculation time.

3. IMPLEMENTATION ISSUES

We have shown that the proposed algorithm offers considerable performance improvement over existing methods, on today's industrial controllers. We will show now that additional benefits can be obtained from its implementation in a parallel architecture, using dedicated processors.

The off-line stage of the algorithm generates a control program composed uniquely of the equivalent to *if...then...else...* statements. Therefore, a dedicated processor, optimized to execute this type of (binary) programs, will permit a considerable improvement in performance over a general purpose processor [16]. Moreover, since each bit of the output torque has its own binary program, nothing prevents all of the k bits to be computed simultaneously. Therefore, we can have k simple processors, one per output bit, computing in parallel all the bits of the output torque, thus achieving a k-fold improvement in performance. This general idea is illustrated in fig. 3.

In fact, the increase in performance is not limited by the number of output bits. The operations involved in the on-line calculations required by the algorithm, allow simple distribution of the computational load amongst any number of processors. The result is an improvement in the on-line computing time proportional to the number of processors. In contrast to other parallel computing systems [14] this improvement is not subject to any theoretical limit and can be achieved without the complex scheduling problems common to other arithmetic manipulator dynamics parallel computing structures [15].

One final comment should be made regarding the applicability of the method to other types of problems. The algorithm is, in fact, not restricted to robot dynamics. It may find application whenever real-time requirements can not be met by existing numerical or analyical computational methods, provided the description of the behavior of the system can be partitioned into tables of acceptable sizes.

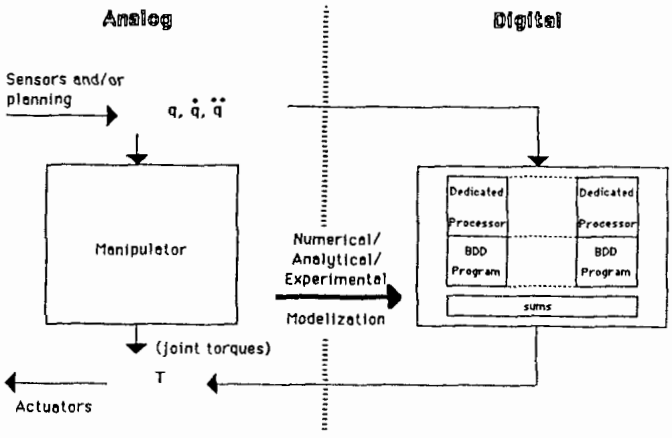


Fig. 3. Parallel computational model with dedicated processors.

4. CONCLUSIONS

In this paper we presented a new computational method for robot manipulator inverse dynamics that achieves a remarkable reduction of the on-line computing time. The algorithm, which is not restricted to manipulator dynamics, leads naturally to, and takes full advantage of, parallel computing structures of very simple, and very fast, dedicated processors.

REFERENCES

[1] J. M. Hollerbach, "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," IEEE Trans. Syst., Man, Cybern., vol. SMC-10, pp. 730-736, Nov., 1980.

[2] J. Y. S. Luh, M. W. Walker and R. P. C. Paul, "On-line Computational Scheme for Mechanical Manipulators," ASME J. Dynamic Syst. Meas. Contr., vol. 102, pp. 69-76, June 1980.

[3] W. M. Silver, "On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators," The Int. J. Robotics Research, vol. 1, no. 2, pp. 60-70, Summer 1982.

[4] M. C. Leu, N. Hemati, "Automated Symbolic Derivation of Dynamic Equations of Motion for Robotic Manipulators," ASME J. Dynamic Syst. Meas. Contr., vol. 108, pp. 172-179, Sept. 1986.

[5] J. Koplic and M. C. Leu, "Computer Generation of Robot Dynamic Equations and the Related Issues," J. of Robotic Systems, vol. 3, no. 3, pp. 301-319, Fall 1986.

[6] D. C. Yang, S. W. Tzeng, "Simplification and Linearization of Manipulator Dynamics by the Design of Inertia Distribution," The Int. J. Robotics Research, vol. 5, no. 3, pp. 120-128, Fall 1986.

[7] C. P. Newman and J. J. Murray, "Customized Computational Robot Dynamics," J. of Robotic Systems, vol. 4, no. 4, pp. 503-526, Aug. 1987.

[8] C. P. Newman and J. J. Murray, "Symbolically Efficient Formulations for Computational Robot Dynamics," J. of Robotic Systems, vol. 4, no. 6, pp. 743-769, Dec. 1987.

[9] J. T. Machado, J. M. Carvalho, J. S. Matos and A. C. Costa , "A Real-Time System for Robot Manipulator Inverse Dynamics Computation," 15th IFAC/IFIP Workshop on Real-Time Programming, Valencia, Spain, May 1988.

[10] J. T. Machado, J. M. Carvalho, A. C. Costa and J. S. Matos, "Robot Manipulator Dynamics -- Towards Better Computational Algorithms," IFAC Symp. on Robot Control, Karlsruhe, FRG, October 1988.

[11] E. J. McCluskey, "Minimization of Boolean Functions," Bell Systems Tech. Journal, 35, no. 5, pp. 1413-1444, Nov. 1956.

[12] B. M. E. Moret, "Decision Trees and Diagrams," Computing Surveys, vol. 14, no. 4, Dec. 1982.

[13] S. B. Akers, "On the Specification and Analysis of Large Digital Functions," in Proc. 7th Int'l. Symp. Fault-Tolerant Comput., pp. 88-93, June 1977.

[14] S. B. Akers, "Binary Decision Diagrams," IEEE Trans. Comput., vol. C-27, no. 6, June 1978.

[15] J. S. Matos, "The Binary Decision Diagram: A Tool for Logic design and Implementation," PhD Dissertation, Syracuse University, Syracuse, N.Y., 1983.

[16] P. J. Zsombsor-Murray, L. J. Vroomen, R. D. Hudson, T. Le-Ngoc and P. H. Holck, "Binary-Decision-Based Programmable Controllers", IEEE Micro, October 1983.