Journal of Intelligent and Robotic Systems **9**: 25–43 1994.
© 1994 *by Kluwer Academic Publishers. Printed in the Netherlands.*

Parallel VLSI Architectures for Real-Time Kinematics of Redundant Robots

IAN D. WALKER and JOSEPH R. CAVALLARO
Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251, U.S.A.

(Received: 25 November 1992)

Abstract. We describe new architectures for the efficient computation of redundant manipulator kinematics (direct and inverse). By calculating the core of the problem in hardware, we can make full use of the redundancy by implementing more complex self-motion algorithms. A key component of our architecture is the calculation in the VLSI hardware of the Singular Value Decomposition of the manipulator Jacobian. Recent advances in VLSI have allowed the mapping of complex algorithms to hardware using systolic arrays with advanced computer arithmetic algorithms, such as the coordinate rotation (CORDIC) algorithms. We use CORDIC arithmetic in the novel design of our special-purpose VLSI array, which is used in computation of the Direct Kinematics Solution (DKS), the manipulator Jacobian, as well as the Jacobian Pseudoinverse. Application-specific (subtask-dependent) portions of the inverse kinematics are handled in parallel by a DSP processor which interfaces with the custom hardware and the host machine. The architecture and algorithm development is valid for general redundant manipulators and a wide range of processors currently available and under development commercially.

Key words. Robot kinematics, VLSI, cordic arithmetic, kinematic redundancy, pseudoinverse.

1. Introduction

The Robot Forward and Inverse Kinematics problems are critical computationally intensive tasks in real time robot control which can benefit from the judicious application of parallel algorithms and architectures [10, 20]. Parallel architectures including systolic arrays are being successfully applied to more basic algorithms in real-time signal processing and image processing, as well as robotics. Previous work on architecture design has centered on robot kinematics, [15, 17, 19, 24, 31, 39] dynamics, [1, 8, 18] and control [3, 13, 27, 37]. Algorithms and architectures for kinematically redundant manipulators have been considered in [6, 22, 34].

Recent advances in VLSI have allowed the mapping of complex algorithms to hardware using systolic arrays with advanced computer arithmetic algorithms, such as the coordinate rotation (CORDIC) algorithms. CORDIC has been used in robotics for inverse kinematics [11, 15, 17] for nonredundant robots and for control [3, 37]. However, it appears that CORDIC has not been utilized in the Direct Kinematics Solution (DKS) and Jacobian computations, where the many rotations present fertile ground for its ability to handle rotations efficiently. In this work, we use CORDIC arithmetic in the novel design of a special-purpose VLSI array, forming a key part of an efficient architecture for redundant robot kinematics computations.

The overall objectives of our research are to design a custom VLSI CORDIC Architecture suitable for redundant manipulator kinematics, to fabricate chips for this architecture, and implement the resulting architecture in our Robotics Laboratory [34]. In this paper, we describe progress in the first stages of this project, the design and implementation of a high performance VLSI array specifically tailored for the robotic application and the combination of the array with a high performance general purpose processor. Digital Signal Processor (DSP) chips present a number of architectural advantages, including high speed arithmetic and communication ports for parallel array construction. The performance advantage of DSP chips over conventional microprocessors has narrowed somewhat with the current generation of Reduced Instruction Set (RISC) floating-point processors. However, the DSP architecture remains important for real-time control and will be the model used in this paper.

The self-motion capability of redundant manipulator designs [23, 28] allow them to significantly outperform conventional current robots, particularly in unstructured environments [33, 35] (for obstacle avoidance, fault tolerance, etc.). However, the increased complexity of solving the inverse kinematics for redundant arms is a key factor in limiting the current practical use of such robots to a fraction of their potential. Typically in applications, inverse kinematics for redundant arms has been 'avoided' by using end effector-based control schemes. However, these schemes do not make provision for the self-motion capability which is the fundamental motivation for redundant joints – indeed in some cases, the resulting joint motion is non-conservative and unpredictable, which is clearly undesired. A number of suggested joint space techniques for self-motion control are based on a well-known algorithm using the Jacobian pseudoinverse [23, 28], which must be computed iteratively in real time in practise.

Here, we restructure and recast the kinematics algorithm to exploit the parallelism in the computation. We then introduce efficient special purpose VLSI arrays to compute the DKS, the Jacobian, and its pseudoinverse. The Singular Value Decomposition (SVD) represents an appealing method of computing the pseudoinverse. Our previous work has shown the applicability of high speed algorithms such as the CORDIC algorithms to the SVD of real matrices [5]. Our architecture features a special purpose array of CORDIC processors to compute the SVD of a matrix. The result is a new and interesting application of design techniques and sheds light on many of the issues involved in applying VLSI methodologies to apparently amorphous technological algorithms. The resulting processor design offers the possibility of implementing in real time some of the more complex schemes for redundant robot kinematic control suggested in the literature.

Our architecture is valid for any type of high performance processor, and is thus not restricted to our particular hardware. We are currently using the Texas Instruments TMS320C30 and TMS320C40 floating-point DSP chips. Based upon the performance of the custom chips designed for the CORDIC SVD array, and utilizing maximum parallelism wherever possible, we estimate that this architecture will compute each iteration for the joint velocities in approximately 400 microseconds. We are currently building custom CMOS chips for this architecture, which is described in more detail in the following.

2. Forward and Inverse Kinematics Algorithms

A key computational bottleneck for controllers is the calculation of inverse kinematics, which produces the time configuration history of the robot joints necessary to produce desired end effector motion. The controller must solve the inverse kinematics problem to produce the correct (joint) trajectories to track.

The problem may be restated as that of solving the direct (forward) kinematics (DKS) equations

$$\underline{x} = f(\underline{\theta}) \tag{1}$$

for the values of the n joint variables $\underline{\theta} \in \Re^n$ given the m end effector position/orientation variables $\underline{x} \in \Re^m$.

Two approaches to inverse kinematics are common for nonredundant robots. One method involves solving (1) directly for the joint angles. Conventionally this involves forming the direct kinematics as a product of 4×4 homogeneous transformation matrices $[H_{i-1}^{i}(\theta_i)]$ where θ_i is the ith component of $\underline{\theta}$,

$$[H_0^n] = [H_0^1(\theta_1)][H_1^2(\theta_2)] \ldots [H_{n-2}^{n-1}(\theta_{n-1})][H_{n-1}^{n}(\theta_n)] \tag{2}$$

and equating with a (desired) end effector transformation $[T_0^n]$. The inverse kinematics is then computed by manipulation of (2) to produce n independent equations for the θ_i's [30]. For many conventional nonredundant robots, (2) yields closed form solutions, and a number of elegant approaches to efficient inverse kinematics computation have been proposed (for example [11, 15, 17]). However, there are in general several solutions to (2) (corresponding to 'elbow-up' 'elbow-down' solutions, etc.) which complicate the solution even for nonredundant arms.

In our architecture, we adopt the 'resolved-rate' approach [38], solving for the joint velocities required for end effector motion, using the manipulator Jacobian. This approach, which can be used for both redundant and nonredundant arms, is more appropriate than solving for $\underline{\theta}$ directly in the case of redundant arms, where there are in general an infinite set of solutions in (1) for $\underline{\theta}$ given $\underline{x}$ and direct solutions to (2) more complex [23, 28].

By differentiation of (1), we obtain the relationship between the joint velocities and end effector velocities expressed as

$$\dot{\underline{x}} = [J]\dot{\underline{\theta}} \tag{3}$$

where the $(m \times n)$ matrix $[J] = \partial f / \partial \underline{\theta}$ is known as the manipulator Jacobian. Equation (3) is used to solve for the joint velocities $\dot{\underline{\theta}}$ and hence the joint variables $\underline{\theta}$ by integration.

Redundant robot inverse kinematic algorithms are typically based on the pseudoinverse of the Jacobian matrix as follows:

$$\underline{\dot{\theta}} = [J^{+}(\underline{\theta})]\underline{\dot{x}} + [I - J^{+}(\underline{\theta})J(\underline{\theta})]\underline{\epsilon} \tag{4}$$

where $[J^{+}]$ is the pseudoinverse of the Jacobian (for more details, see [23, 28]). In (4), I is the $(n \times n)$ identity matrix, and $\underline{\epsilon}$ is an $(n \times 1)$ column vector whose values may be arbitrarily selected [23, 28]. Conventional nonredundant manipulators have $m = n$, in which case there is a unique solution for $\underline{\dot{\theta}}$, ($[J^{+}] = [J^{-1}]$ in this case, and $[I - J^{+}J] = [0]$). Redundant arms have n larger than m (extra joints), and different choices for $\underline{\epsilon}$ determine different possible arm postures, each of which will allow the specified end effector motion $\underline{\dot{x}}$. This capability is however gained at the expense of significantly increased complexity via the need to compute the pseudoinverse and the second term in (4). Earlier we proposed an architecture which computes both the direct kinematics solution (2) and the manipulator Jacobian using a parallel array. The architecture then computes the joint velocities using (4) to complete the kinematics.

3. Novel VLSI CORDIC/DSP Architecture

Figure 1 presents a conventional computing environment for the inverse kinematics calculation. We propose an Inverse Kinematics Engine (IKE) architecture, which incorporates a special-purpose VLSI array and a programmable DSP processor. Figure 2 shows the proposed Inverse Kinematic Engine (IKE) which is a key part of the architecture presented in this paper. The IKE reduces the computational load on the Real Time Controller through the use of custom VLSI hardware and DSP processors. Figure 3 shows the enhanced VLSI CORDIC/DSP architecture demonstrating the Inverse Kinematic Engine (IKE) configuration of Figure 2. The

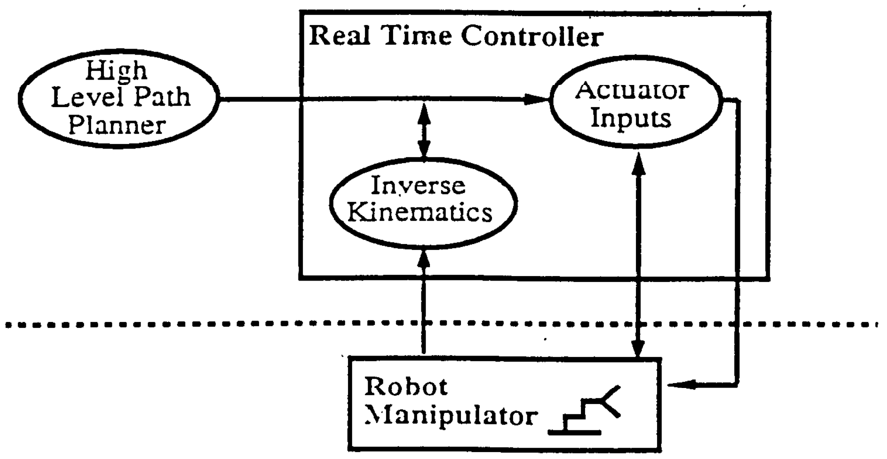


Fig. 1. Robot Control using internal inverse kinematics calculation.

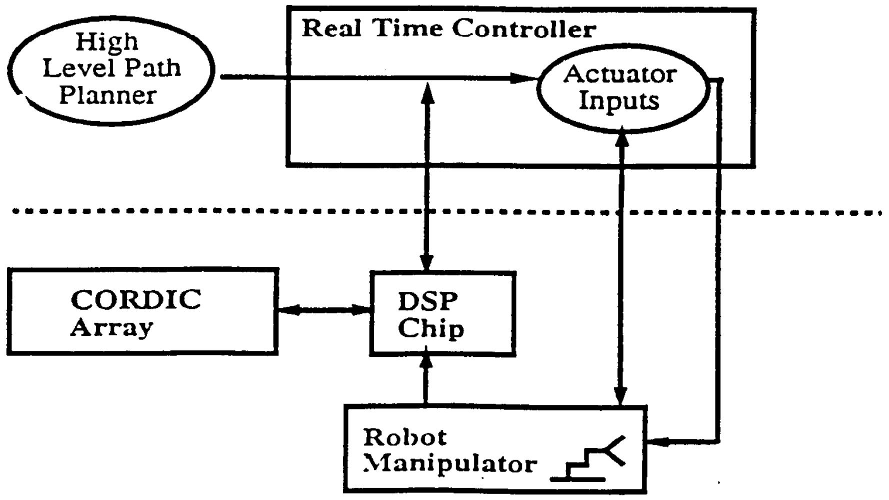


Fig. 2. Robot control using high speed CORDIC IKE.

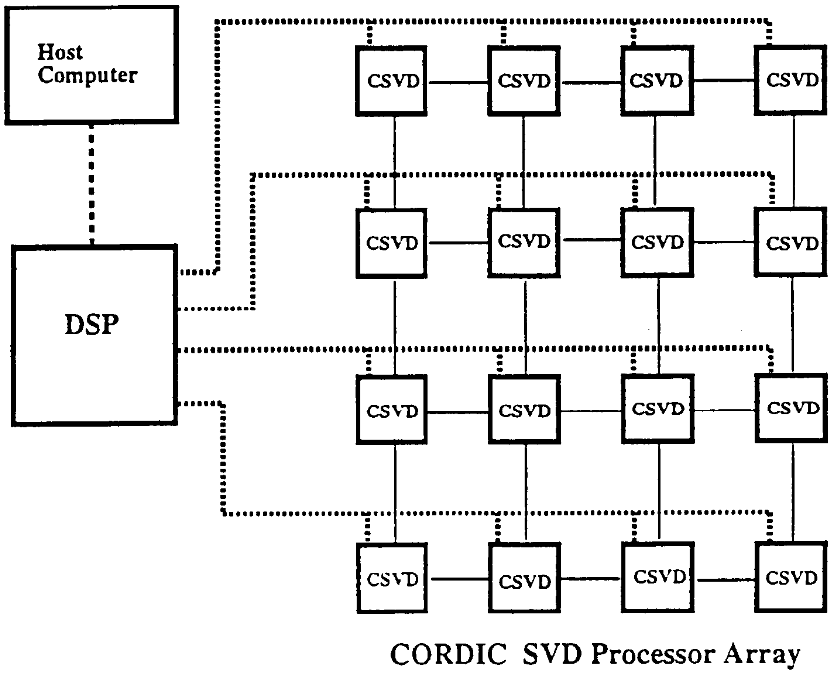


Fig. 3. DSP – CORDIC array configuration.

interconnection of the hardware components in the proposed IKE is shown. The CORDIC SVD array acts as a co-processor for the general-purpose DSP processor as shown in Figure 3. In a particular implementation, the Processor Element could be a DSP chip, such as the Texas Instruments TMS320C30 [29] or TMS320C40 [7], or one of the INMOS Transputer chips [14].

In our architecture, both the DKS and the Jacobian are calculated using the CORDIC array and the DSP chip. To perform the inverse kinematics, the pseudoinverse of the Jacobian is performed in hardware using the CORDIC SVD array. The pseudoinverse is used to calculate the next iteration of joint velocities, with the necessary matrix-matrix and matrix-vector computations carried out by the DSP chip using the information emanating from the CORDIC SVD array. As part of our architecture, we can calculate the DKS in parallel with the Inverse Kinematics. The high level Overall Kinematics algorithm is structured to exploit the processing capabilities of VLSI CORDIC/DSP arrays.

Our architecture has similarities with that of [3, 37] in which a general purpose robotics processor with two CORDIC subunits was devised. However, our special-purpose CORDIC array with multiple CORDIC units allows us more flexibility in partitioning the various portions of the algorithm in parallel. Our architecture is also more general than that proposed in [13], which is specific to nonredundant robots, and [22], which does not have the CORDIC SVD array or efficient parallel Jacobian calculation. Our approach is valid for more general robots than that in [17], which used a pipelined CORDIC architecture for closed form nonredundant robot inverse kinematics.

This work concentrates on the design of the first level building blocks, including the DKS, Jacobian, Jacobian SVD and pseudoinverse sections.

4. New Parallel DKS and Jacobian Using CORDIC

We use our CORDIC architecture to take advantage of both the parallelism and the large number of rotations inherent in the DKS and Jacobian calculations. We adopt the conventional Jacobian formulation expressed in a base coordinate frame [30]. Other work [24, 25] has considered Jacobian models expressed at varying coordinate systems. Our architecture can easily be used to compute any of these Jacobians. However, the Jacobian in base coordinates is standard in the literature, thus is a good reference case for our architecture, and will be adopted in the following. For the purposes of demonstration, we use the example of an eight revolute joint arm. However, our architecture is not limited in the number or type of joints in the robot.

4.1. ROTATIONS USING CORDIC ARITHMETIC

The Coordinate Rotation Digital Computer (CORDIC) algorithms provide a fast hardware method to calculate vector rotations and inverse tangents. These

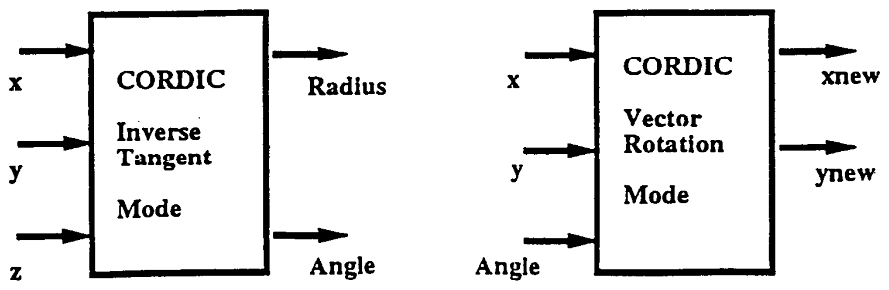


Fig. 4. CORDIC arithmetic processor elements.

algorithms were initially described by Volder [32] and further extended to hyperbolic functions by Walther [36].

The CORDIC iteration equations which are to be implemented in hardware are:

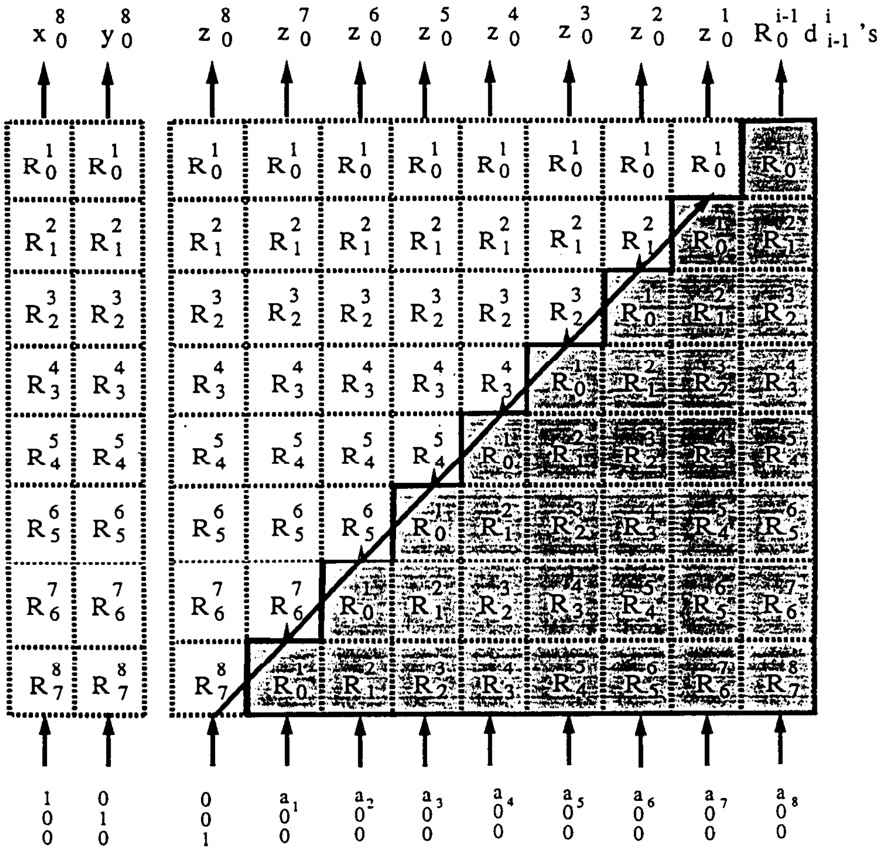
$$\begin{aligned} x_{i+1} &= x_i + \delta_i y_i 2^{-i}, \\ y_{i+1} &= y_i - \delta_i x_i 2^{-i}, \\ z_{i+1} &= z_i + \delta_i \theta_i. \end{aligned} \tag{5}$$

The variable, z_i, contains the total rotation angle applied, θ_i is the current rotation angle increment, and $\delta_i = \pm 1$. Through the appropriate selection of each δ_i, either the initial z_0 value can be reduced to zero (vector rotation) or the initial y_0 value can be reduced to zero (inverse tangent). These equations can be implemented with simple structures: registers, shifters, adders, and a small ROM. Figure 4 presents a block diagram of the CORDIC processor. The CORDIC algorithms have been applied by Cavallaro and Luk [5] to the Jacobi method for the SVD to produce a processor architecture which is currently being implemented by us [4, 12] at Rice University. For the current eight joint robot architecture, we require an array to compute an 8×8 SVD. This can be performed on a 4×4 array of CORDIC SVD processors where each unit operates on a 2×2 sub-matrix. Additionally, at each iteration, prior to taking the SVD, individual CORDIC processors within the array are used (as described in the following section, see Figure 5) to compute rotations required for the DKS and Jacobian. The operation of each CORDIC unit and data flow between it and the DSP chip are shown in Figure 6.

4.2. DSP/PARALLEL CORDIC JACOBIAN FORMATION

At each iteration, the values of the 8 joint variables $\underline{\theta}$ are streamed in parallel to generate the following n matrices $[R^i_{i-1}]$ and n vectors $\underline{d}^i_{i-1}$:

$$[R^i_{i-1}] = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 \\ -\sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_i & \sin\alpha_i \\ 0 & -\sin\alpha_i & \cos\alpha_i \end{bmatrix}, \tag{6}$$



x_0^8 y_0^8 z_0^8 z_0^7 z_0^6 z_0^5 z_0^4 z_0^3 z_0^2 z_0^1 $R_0^{i-1}\,d_{i-1}^{i}$'s
R_0^1 R_1^2 R_2^3 R_3^4 R_4^5 R_5^6 R_6^7 R_7^8
R_0^1 R_0^2 R_1^2 R_0^3 R_1^3 R_2^3 R_0^4 R_1^4 R_2^4 R_3^4 R_0^5 R_1^5 R_2^5 R_3^5 R_4^5 R_0^6 R_1^6 R_2^6 R_3^6 R_4^6 R_5^6 R_0^7 R_1^7 R_2^7 R_3^7 R_4^7 R_5^7 R_6^7 R_0^8 R_1^8 R_2^8 R_3^8 R_4^8 R_5^8 R_6^8 R_7^8
1 0 0 | 0 1 0 | 0 0 1 | a_1 0^1 0 | a_2 0^2 0 | a_3 0^3 0 | a_4 0^4 0 | a_5 0^5 0 | a_6 0^6 0 | a_7 0^7 0 | a_8 0^8 0

Fig. 5. Dataflow for Jacobian matrix formation using CORDIC rotators.

$$\underline{d}_{i-1}^{i} = \begin{bmatrix} a_i \cos\theta_i \\ a_i \sin\theta_i \\ d_i \end{bmatrix}. \tag{7}$$

In the above, α_i, a_i, and d_i are constants related to the dimensions of the robot. These quantities must then be used to form the Jacobian $[J]$, which takes the form (assuming revolute joints) [30]:

$$[J] = \begin{bmatrix} \underline{z}_0^0 \times \underline{d}_0^8 \cdots \underline{z}_0^i \times (\underline{d}_0^8 - \underline{d}_0^i) \cdots \underline{z}_0^{n-1} \times (\underline{d}_0^8 - \underline{d}_0^{n-1}) \\ \underline{z}_0^0 \quad \cdots \quad \underline{z}_0^i \quad \cdots \quad \underline{z}_0^{n-1} \end{bmatrix}, \tag{8}$$

where $\times$ denotes the cross product operation. The elements in the Jacobian are

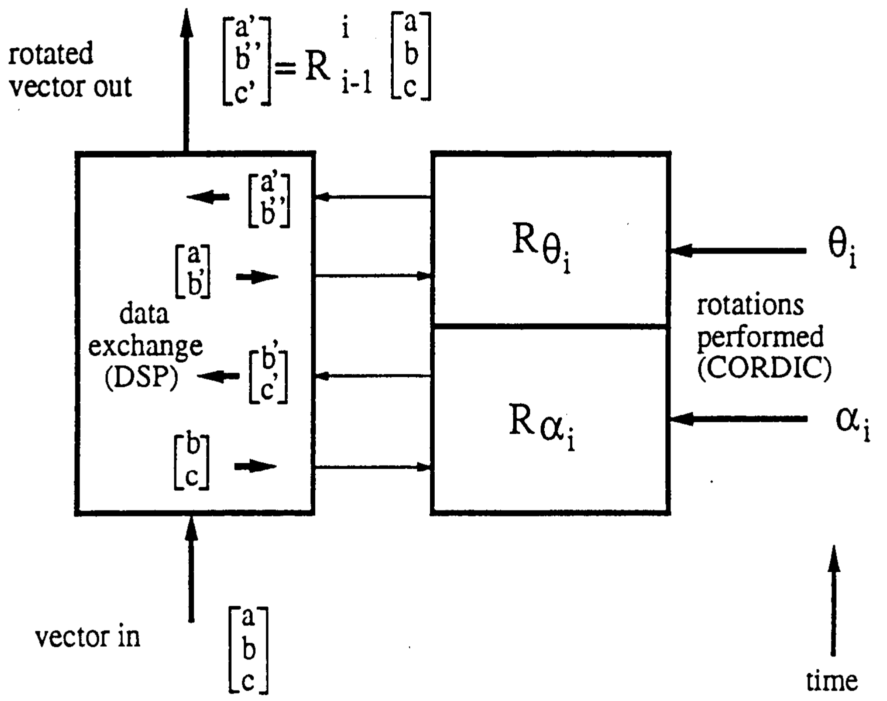


Fig. 6. CORDIC co-processor organization for Jacobian calculation.

formed recursively from the above quantities $\underline{d}^i_{i-1}$ and $[R^i_{i-1}]$ as follows:

$$[R^i_0] = [R^{i-1}_0][R^i_{i-1}], \tag{9}$$

$$\underline{d}^i_0 = \underline{d}^{i-1}_0 + [R^{i-1}_0]\underline{d}^i_{i-1}, \tag{10}$$

for $i = 1, \ldots, 8$ where the initial conditions are $\underline{d}^0_0 = \underline{0}$ and $[R^0_0] = [I]$ with $[I]$ the 3×3 identity matrix and the $\underline{z}^i_0$ vectors are the third columns of the $[R^i_0]$ matrices.

A CORDIC array to compute the Jacobian is shown in Figure 5. This logical array is formed using physical sub-units of our special-purpose CORDIC SVD array. From (8), the top and bottom halves of J are formed differently. Both require the computation of (9) in the double triangular array shown in Figure 5. The d vectors required for the upper portion of the Jacobian are streamed in the lower triangular array. The physical array is composed of nine CORDIC rotation modules operating in parallel, with the initial seeds shown at the bottom of the array. The loading and unloading/shifting of data is performed by the DSP Processor. A diagram of the rotations produced by each module, with the data exchanges performed by the DSP processor, is shown in Figure 6. The total propagation time for this task will be 16 CORDIC rotation cycles using nine CORDIC processors in parallel. CORDIC scale factor correction will also be performed as the rotations are being applied. Notice that we have full utilization of the array.

When we seed the uppe. triangular array with $[0,0,1]^T$ then we generate the z's. Similarly, Equation (10) is seeded in the lower triangular array by d's in (7) (notice we need only seed by $[a_i,0,0]^T$ to produce the d_{i-1}^i's in one rotation, with a sign change in the DSP). Notice that by performing a succession of 2×2 rotations and holding the third vector element in register, we make a computational savings in the matrix rotations (we do not explicitly perform the two multiplications by 1 in (6)).

The subtractions and cross products to form the final Jacobian are formed by the DSP processor. The appropriate columns are then collected to form the Jacobian matrix. Other methods have been proposed to compute the Jacobian [13, 21, 24, 25, 39]. Our approach adds to this body of techniques, using CORDIC for the first time, and making full use of the rotations inherent in the algorithm.

In parallel with the formulation of the Jacobian, the remainder of the direct kinematics solution (x_0^8 and y_0^8) is performed by appropriate rotations of $[1,0,0]^T$ and $[0,1,0]^T$ in two other CORDIC units (shown on the left in Figure 5). When coupled with z_0^8 and d_0^8 from the Jacobian calculation, the entire direct kinematics is formed in parallel with the Jacobian calculations (recall that

$$[H_0^8] = \begin{bmatrix} x_0^8 & y_0^8 & z_0^8 & d_0^8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

in (2)). Notice that the total direct kinematics solution is performed using four CORDIC units, with the Jacobian calculations occupying nine units. The decomposition depicted above is not the only possible method. For example, we could perform the Jacobian operation using eight modules in parallel (shifting the triangular portions). However, in this case, the operation would take nine steps in Figure 5 instead of eight. Previous architectures to compute the DKS [13, 19] did not use CORDIC to compute the rotations. By using the CORDIC units, we are making full use of our special-purpose CORDIC SVD array.

5. Jacobian Pseudoinverse Calculation

Given the Jacobian, we next need to compute its pseudoinverse to compute the joint velocities via (4). In general, the most efficient means of computing a pseudoinverse is by its Singular Value Decomposition [16]. Previous work in computing the pseudoinverse did not use the SVD or CORDIC [6, 22]. In the following, we review the SVD, and note how the application of CORDIC results in an efficient special-purpose parallel processor array.

5.1. SINGULAR VALUE DECOMPOSITION USING CORDIC

The Singular Value Decomposition (SVD) represents an appealing method of computing the pseudoinverse. The SVD is more closely mapped onto hardware through the use of the CORDIC algorithms. In this paper, we build upon our previous work on the application of CORDIC arithmetic to the SVD, by taking advantage of the features offered by the CORDIC SVD in efficiently mapping algorithms onto hardware.

The singular value decomposition [9] of a $p \times p$ matrix M is

$$M = U\Sigma V^{\mathrm{T}}, \tag{12}$$

where U and V are orthogonal matrices and Σ is a diagonal matrix of singular values. The SVD is a computationally complex algorithm which benefits from a VLSI parallel array.

The systolic array of Brent, Luk, and Van Loan [2] uses a square array of processors to implement the parallel Jacobi Method for the SVD. Figure 7 shows a typical array architecture to perform a 6×6 SVD. In the Brent-Luk-Van Loan array, the matrix is divided into 2×2 submatrices. Each processor element contains a 2×2 submatrix. The array architecture is scalable. There are two types of data flowing in this array. Rotation angles generated by the diagonal processors flow systolically along the rows and columns of the array. Matrix data elements are exchanged diagonally, after the diagonal neighbor has received and applied the

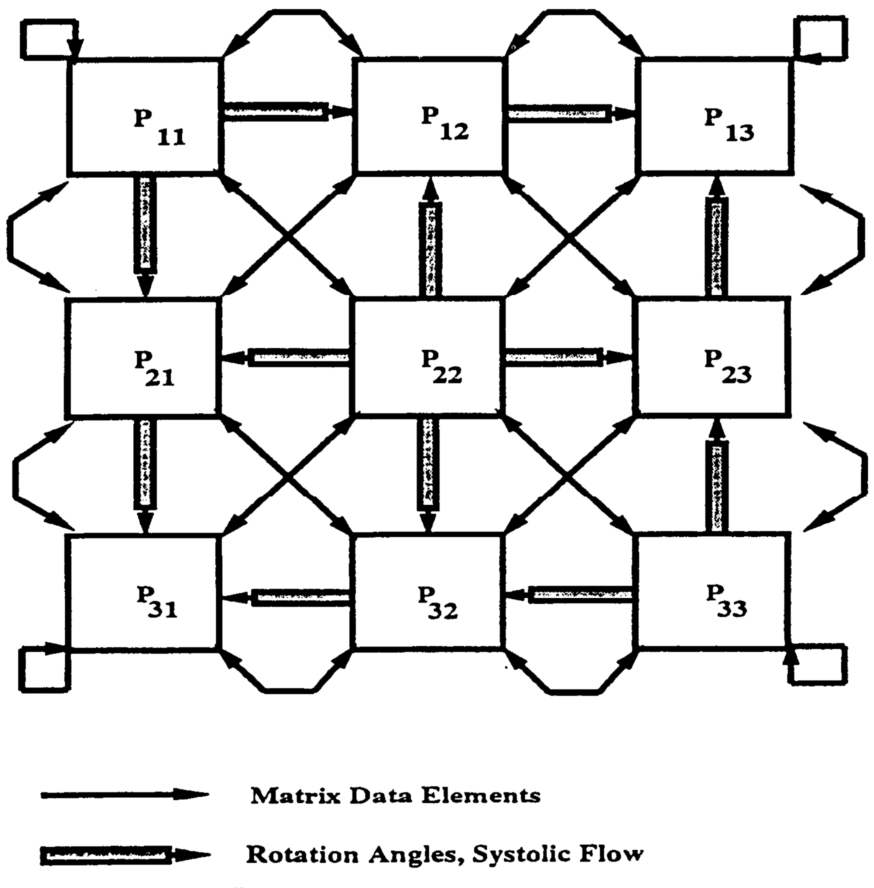


Fig. 7. Brent Luk Van Loan array for the SVD.

necessary rotation angles. This leads to 'waves' of activity moving diagonally away from the main array diagonal.

A 2×2 SVD can be described as

$$R(\theta_l)^{\mathrm{T}} \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix}, \tag{13}$$

where θ_l and θ_r are the left and right rotation angles, respectively. The rotation matrix is

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}, \tag{14}$$

and the input matrix is

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}. \tag{15}$$

In a typical computer algorithm for the SVD, the sines and cosines of the rotation angles are computed through formulas that require division and square root operations. The explicit angles are not required and only the sines and cosines are computed. The rotations are then applied to the 2×2 submatrix using standard matrix multiplication techniques. However, time-consuming operations such as multiplication, division, and square root are needed.

As mentioned above, the CORDIC algorithms provide a fast hardware method to calculate vector rotations and inverse tangents which are essential operations for the SVD. We integrate the CORDIC SVD processor array with a DSP processor to compute inverse kinematics for redundant robots, a more general problem going beyond the earlier work on CORDIC for non-redundant robots [11, 15, 17].

5.2. FORMATION OF PSEUDOINVERSE FROM SVD

The pseudoinverse is easily expressed in terms of the information resulting from the SVD as follows [16]:

$$J^+ = V\Sigma^+ U^T, \tag{16}$$

where U and V are defined from the SVD, and Σ^+ is a $n \times m$ matrix with reciprocals of the singular values in Σ along the leading diagonal, and zeros elsewhere. The singular values of the Jacobian are obtained from the CORDIC SVD array. The current CORDIC SVD processor can be enhanced to collect the matrices U and V. The reciprocals of the singular values required for Σ^+ are generated from Σ by the DSP chip. Finally, the product $J^+ = V\Sigma^+ U^T$ is formed in the DSP chip.

5.3. STRUCTURE OF INVERSE KINEMATIC ALGORITHM

The algorithm to be computed becomes, when discretized,

$$\dot{\underline{\theta}}_t = [J^+(\underline{\theta}_{t-1})]\dot{\underline{x}}_t + [I - J^+(\underline{\theta}_{t-1})J(\underline{\theta}_{t-1})]\underline{\epsilon}_t. \tag{17}$$

The quantity $\underline{\theta}_{t-1}$ is assumed available via sensory information, and $\underline{\dot{x}}_t$ given from off-line end effector planning. We see that at each stage, the quantities that must be obtained are: first, the updated version of the Jacobian; second, the calculation of the Jacobian Pseudoinverse; and third, the calculation of the new value of $\underline{\dot{\theta}}_t$ using (17). The algorithm is configured as follows:

```
Algorithm VLSI CORDIC Parallel Inverse Kinematics()
begin;
    compute Jacobian;
    compute SVD;
    do (Pseudoinverse, Null Space Projector);
    do ($[J^{+}]\underline{\dot{x}}$, $[I - J^{+}J]\underline{\epsilon}$);
    do ($\underline{\dot{\theta}} = [J^{+}]\underline{\dot{x}} + [I - J^{+}J]\underline{\epsilon}$);
    output $\underline{\dot{\theta}}$;
end;
```

The value of $\underline{\epsilon}_t$ depends on the particular redundancy resolution scheme selected, and is application dependent. Part of our ongoing research concerns calculation of different schemes for selecting $\underline{\epsilon}_t$ within the DSP processor. This is possible due to the ability of DSP chips to handle differing algorithms (notice that most schemes for selecting $\underline{\epsilon}_t$ require values of $\underline{\theta}_{t-1}$ and/or $\underline{\dot{\theta}}_t$, which are available locally in our architecture). This allows the entire inverse kinematics solution to be contained within the Inverse Kinematics Engine, and separate from the host machine. The key to our architecture is the calculation of the rotations making up the Jacobian and direct kinematics using CORDIC processors.

5.4. GENERATION OF NEW $\underline{\dot{\theta}}_t$'S AND $\underline{\theta}_t$'S

After the completion of the SVD, several steps which involve a series of matrix-vector multiplications occur in (4). These are implemented using the DSP processor. The top portion of Figure 8 shows the necessary operations. The matrix-vector multiplications necessary to generate the Null Space projector will be performed by the DSP processor along with the Pseudoinverse computation.

The DSP processor also forms the integration required to produce the joint angles from the joint velocities. The programmability of the DSP processor allows the use of different integration routines, such as the Adams predictor-corrector [13], Runge–Kutta methods, etc., to accuracy as desired for particular applications.

6. VLSI Implementation Issues

In this section, we investigate the issues involved in designing special purpose architectures to allow exploitation of the structure of the manipulator inverse kinematics algorithms. In this architecture, the CORDIC co-processor array is utilized for a number of functions as shown in Figure 9. Although composed of special-purpose

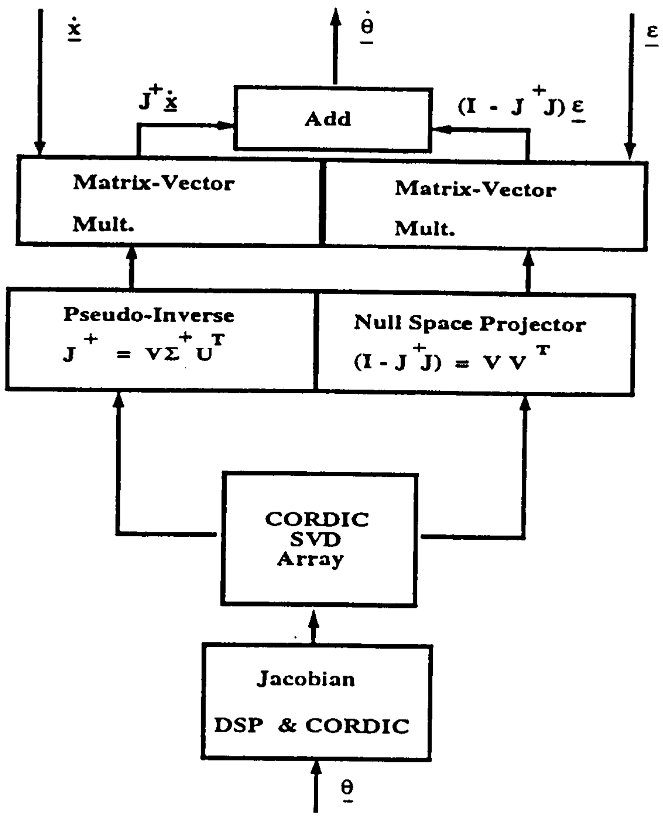


Fig. 8. CORDIC Inverse Kinematics Engine (IKE) algorithm.

chips, each CORDIC SVD chip (CSVD) can act as a simple CORDIC co-processor unit or as part of the systolic array for the SVD. Under the control of the DSP chip, the 9 CSVD chips in the lower righthand corner of Figure 9 compute parameters for the Jacobian matrix. Two of these nine plus two additional CSVD chips in parallel compute the DKS. At a later point in the inverse kinematics algorithm, all 16 CSVD chips compute the SVD of the Jacobian.

6.1. CORDIC ARRAY PROCESSOR ELEMENT

The CORDIC SVD processor is implemented in a 2 μm CMOS technology and fabricated through the MOSIS service. The chip measures 6900×5600 μm and contains over 26 000 transistors [12]. The chip is primarily designed to implement the 2×2 SVD systolic algorithm, but performs in several different modes in response to a control word. The chip is also capable of acting as a dedicated

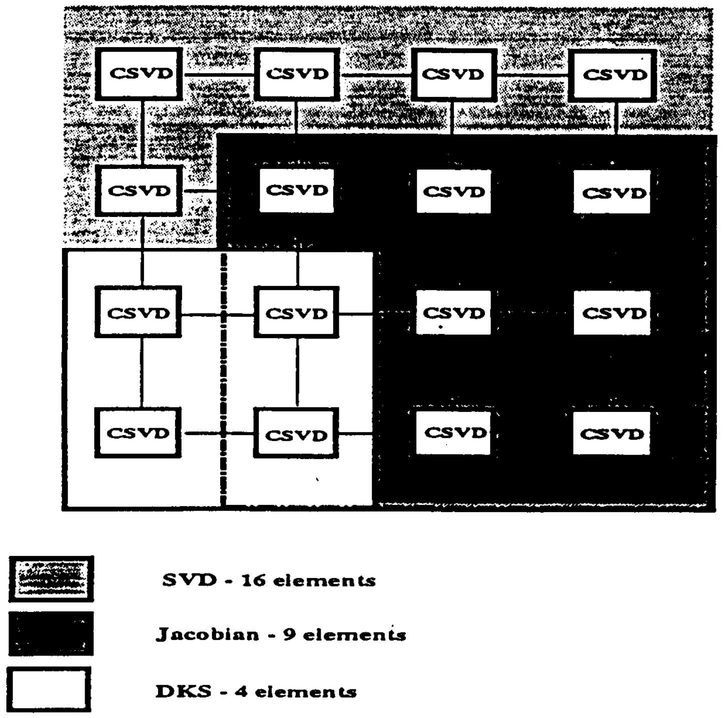


Fig. 9. Utilization of Processor Elements of CORDIC SVD array.

CORDIC co-processor and in fact contains two 16-bit fixed-point CORDIC units. This first generation CSVD chip uses a conservative two-phase nonoverlapping clocking strategy and is fully functional at 2 MHz. The design uses custom adder and barrel shifter cells designed with the *magic* layout editor. The control units are implemented as PLAs and the higher level routing was completed with *octtools*.

We are currently designing a second generation chip which will reduce the delays in the 16-bit barrel shifter section of the CORDIC units. We expect to improve the performance of the chip so that it would operate at 10 MHz. The CSVD chip communication ports are also being modified to interconnect with the TMS320C40 8 bit communication ports.

6.2. DIGITAL SIGNAL PROCESSORS

The current generation of floating-point DSP chips combine the flexibility of a general purpose microprocessor with high performance arithmetic and communication facilities. The TMS320C30 chip has a 60 ns processor cycle time and hardware support for single cycle floating-point multiplication and addition. This leads to a high speed multiply and accumulate operation important for many of the matrix products needed for inverse kinematics. DSP chips are highly integrated and contain on-board RAM. We have been using a TMS320C30 evaluation module board as part of our implementation. The current generation TMS320C40 has a 40 ns processor cycle time and six hardware communication ports to support multiprocessing [7] and will provide a parallel interface to the CORDIC SVD array. We are currently working with a simulator for the TMS320C40, and projected performance results of our robotic algorithms using the TMS320C40 based on the simulator are presented here. The Texas Instruments DSP chips can be programmed in a high-level language, such as 'C', and have real-time operating system software.

6.3. PERFORMANCE ESTIMATES

An inverse kinematics algorithm for a PUMA robot augmented with a seventh joint was implemented in the 'C' programming language. The code was compiled and loaded to maximize the use of the on-chip RAM, the zero wait-state local and global memory, the on-chip instruction cache, and the register-argument function model. The program was benchmarked on the TMS320C40 simulator and the results for each of the major subroutines and the total time are presented in Table I. The table also contains data for the AT&T DSP32 chip presented by Maciejewski [22] using an updating scheme for the SVD. Our code computes the complete SVD using a Golub–Reinsch algorithm for the serial case on the TMS320C40 and a Jacobi based algorithm for the CORDIC SVD array.

The performance of the algorithm could be improved by replacing the matrix multiplication and transpose routines with assembly language subroutines. For instance, multiplication of two 7×7 matrices requires 2500 TMS320C40 processor cycles or 100 μs using a 'C' language algorithm. Optimized assembly language code for the same problem would require approximately 672 cycles or 27 μs. Further speedup could be achieved by using a parallel array of TMS320C40 chips [26].

In our proposed architecture, we plan to replace the SVD subroutine which consumes over 70% of the execution time of the 'C' language inverse kinematics algorithm with the custom VLSI CORDIC SVD array. This array requires time for loading, computation, flushing, and unloading. The SVD algorithm requires $O(\log n)$ computation sweeps for convergence. For the 8×8 SVD, five sweeps will guarantee convergence. Based on the internal cycle counts, it will take 2560 cycles

Table I. Performance of Inverse Kinematics Algorithm.

Major subroutines	AT&T DSP32 Ref. [22] serial	TMS320C40 Initial 'C' serial	AT&T DSP32 Ref. [22] 4 parallel	TMS320C40 assembly 1st. gen. CSVD array	TMS320C40 assembly 2nd. gen. CSVD array
SVD	6.2 ms	1.3502 ms	1.7 ms	1.28 ms	256 μs
Pseudo-inverse	0.5 ms	0.3234 ms	0.2 ms	135 μs	135 μs
Total	6.7 ms	1.6736 ms	1.9 ms	1.45 ms	400 μs

to compute the SVD. The actual time can be found by multiplying the cycle count by the reciprocal of the maximum frequency. An array of 16 first generation CSVD chips which operate at 2 MHz will therefore compute the SVD in 1.28 ms. The second generation CSVD chip should achieve an increase in clock frequency to 10 MHz and compute the SVD of the Jacobian in 256 μs. Even higher performance is possible if the chip were to be fabricated in the recently announced MOSIS 0.8 μm n-well CMOS process. The cost of the new process is somewhat beyond the range of academic proof of concept implementations.

As a final performance estimate, we consider an array of second generation CSVD chips and the TMS320C40 programmed in assembly language. From (17) and Figure 8, the work which remains beyond the SVD is the reciprocal of the singular values to form Σ^+, three matrix multiplications, two matrix vector multiplications, and a vector addition. This can be conservatively approximated by the time for five matrix multiplications. Using the above estimate of 27 μs for assembly language matrix multiplication, yields 135 μs. The total time including the SVD for a seven joint robot will therefore be 400 μs.

7. Conclusions

In this paper, we describe and detail a new architecture for efficiently computing the kinematic equations (direct and inverse) for kinematically redundant robots. The architecture combines a CORDIC processor array and programmable DSP processors in a novel and interesting manner, fusing together several systolic algorithms. We detail a new parallel method to compute both the Direct Kinematics Solution and the manipulator Jacobian in parallel using units of the CORDIC array. The array, which acts as a co-processor to the DSP chip, is then used to compute the SVD of the Jacobian and its pseudoinverse. The DSP processor is used to compute the joint velocities and positions to complete the inverse kinematics solution. The architecture is valid for nonredundant as well as redundant arms, and can use any type of high performance DSP chip. We have constructed the first generation CORDIC SVD chip and are currently building, testing, and integrating the CORDIC SVD array with the Texas Instruments TMS320C40 processor. We estimate the time for each iteration of the inverse kinematics of an eight joint arm using our architecture to be 400 μs. This suggests that some of the more complex schemes for redundancy resolution suggested in the literature can become tractable in real-time using our architecture.

Acknowledgements

This work was supported in part by the National Science Foundation under grants MIP-8909498 and MSS-9024391 and by DOE Sandia National Laboratory Contract #18-4379A. The authors would like to thank Dr. William Gordon, G. Scott Thomas and Russell H. Price of the Digital Signal Processing Division of Texas Instruments, Houston for assistance with the TMS320 DSP processor.

References

1. Amin-Javaheri, M. and Orin, D. E., Systolic architectures for the manipulator inertia matrix, *IEEE Trans. Systems Man, Cybernet.* **18**(6), 939–951 (1988).
2. Brent, R. P., Luk, F. T. and Van Loan, C. F., Computation of the singular value decomposition using mesh-connected processors, *J. VLSI Comput. Systems* **1**(3), 242–270 (1985).
3. Butner, S. E., Wang, Y., Mangaser, A. and Jordan, S., Design and simulation of RIPS: An advanced robot control system, in *Proc. 1988 IEEE Conf. Robotics and Automation*, Philadelphia, PA, 1988, pp. 470–474.
4. Cavallaro, J. R., Keleher, M. P., Price, R. H. and Thomas, G. S., VLSI implementation of a CORDIC SVD processor, *Proc. 8th Biennial University/Government/Industry Microelectronics Symposium*, June 1989, pp. 256–260.
5. Cavallaro, J. R. and Luk, F. T., CORDIC Arithmetic for an SVD processor, *J. of Parallel and Distributed Computing* **5**(3), 271–290 (1988).
6. Chang, P. R. and Lee, C. S. G., Residue arithmetic VLSI array architecture for manipulator pseudo-inverse Jacobian computation, *IEEE Trans. Robot. Automat.* **5**(5), 569–582 (1989).
7. Chen, D. C. and Price, R. H., A real-time TMS320C40 based parallel system for high rate digital signal processing, in *Proc. 1991 IEEE ICASSP*, Toronto, Canada, 1991, pp. 1573–1576.
8. Fijany, A. and Bejczy, A. K., A class of parallel algorithms for computation of the manipulator inertia matrix. *IEEE Trans. Robot. Automat.* **5**(5), 600–615 (1989).
9. Golub, G. H. and Van Loan, C. F., *Matrix Computations*, 2nd edn, Johns Hopkins Univ. Press, Baltimore, MD, 1989.
10. Graham, J. H., Special computer architectures for robotics: Tutorial and survey. *IEEE Trans. Robot. Automat.* **5**(5), 543–554 (1989).
11. Harber, R. G., Li, J., Hu, X. and Bass, S. C., The application of bit-serial CORDIC computational units to the design of inverse kinematics processors, in *Proc. 1988 IEEE Conf. Robotics and Automation*, Philadelphia, PA, 1988, pp. 1152–1157.
12. Hemkumar, N. D., Kota, K. and Cavallaro, J. R., CAPE – VLSI implementation of a systolic processor array: architecture, design and testing, *Proc. 9th IEEE Biennial University/Government/Industry Microelectronics Symposium*, June 1991, pp. 64–69.
13. Hsia, T. C., Current, K. W., Mao, Z., Chu, W. S., Liu, J., Lu, G. Z. and Han, W. H., A proposed new VLSI architecture for real-time robot manipulator control, *Int. J. Robot. Automat.* **6**(4), 169–178 (1991).
14. Jones, D. I. and Fleming, P. J., Control applications of transputers, in P. J. Fleming (ed), *Parallel Processing in Control: the Transputer and Other Architectures*, Peter Peregrinus Ltd., London, 1988, pp. 101–125.
15. Kameyama, M., Matsumoto, T., Egami, H. and Higuchi, T., Implementation of a high performance LSI for inverse kinematics computation, in *Proc. 1989 IEEE Conf. Robotics and Automation*, Scottsdale, AZ, 1988, pp. 757–762.
16. Klema, V. C. and Laub, A. J., The singular value decomposition: Its computation and some applications, *IEEE Trans. Automat. Control* **AC-25**(2), 164–176 (1980).
17. Lee, C. S. G. and Chang, P. R., A maximum pipelined CORDIC architecture for inverse kinematic position computation, *IEEE J. Robot. Automat.* **RA-3**(5), 445–458 (1987).
18. Lee, C. S. G. and Chen, C. L., Efficient mapping algorithms for scheduling robot inverse dynamics computation on a multiprocessor system, *IEEE Trans. Systems Man Cybernet.* **20**(3), 582–595 (1990).
19. Leung, S. S. and Shanblatt, M. A., Real-time DKS on a single chip, *IEEE Trans. Robot. Automat.* 3(4), 281–290 (1987).
20. Leung, S. S. and Shanblatt, M. A., Computer architecture design for robotics, in *Proc. 1988 IEEE Conf. Robotics and Automation*, Philadelphia, PA, 1988, pp. 453–456.
21. Li, C. J., Hemami, A. and Sankar, T. S., An efficient computational method of the Jacobian for robot manipulators, *Robotica* **9**, 231–234 (1991).

22. Maciejewski, A. A. and Reagin, J. M., A parallel algorithm and architecture for the control of kinematically redundant manipulators, in *Proc. 1992 IEEE Conf. Robotics and Automation*, Nice, France, 1992, pp. 488–493.
23. Nenchev, D. N., Redundancy resolution through local optimization: A review. *J. Robotic Systems* **6**(6), 769–798 (1989).
24. Orin, D. E., Olson, K. W. and Chao, H. H., Systolic architectures for computation of the Jacobian for robot manipulators, in J. H. Graham (ed), *Computer Architectures for Robotics and Automation*. Gordon and Breach, London, 1987, pp. 39–67.
25. Orin, D. E. and Schrader, W. W., Efficient computation of the Jacobian for robot manipulators, *Int. J. Robot. Res.* **3**(4), 66–75 (1984).
26. Piedra, R. M., A parallel approach for solving matrix multiplication on the TMS320C4x DSP, Technical report, Texas Instruments DSP Application Report, Houston, TX, 1991.
27. Sadayappan, P., Ling, Y. L. C., Olson, K. W. and Orin, D. E., A restructurable VLSI robotics vector processor architecture for real-time control, *IEEE Trans. Robot. Automat.* **5**(5), 583–599 (1989).
28. Siciliano, B., Kinematic control of redundant robot manipulators: A tutorial, *J. Intelligent and Robotic Systems* **3**(3), 201–210 (1990).
29. Smiarowski, A. Jr. and Anderson, J. N., A fast computer architecture for the control of robots, *Comput. Elec. Eng.* **17**(3), 217–235 (1991).
30. Spong, M. W. and Vidyasagar, M., *Robot Dynamics and Control*, Wiley, New York, 1989.
31. Tourassis, V. D. and Ang, M. H. Jr., A modular architecture for inverse robot kinematics, *IEEE Trans. Robot. Automat.* **5**(5), 555–568 (1989).
32. Volder, J., The CORDIC trigonometric computing technique, *IRE Trans. Elec. Comput.* **EC-8**(3), 330–334 (1959).
33. Walker, I. D., The use of kinematic redundancy in reducing impact and contact effects in manipulation, in *Proc. 1990 IEEE Conf. Robotics and Automation*, Cincinnati, OH, 1990, pp. 434–439.
34. Walker, I. D. and Cavallaro, J. R., Parallel VLSI architectures for real-time control of redundant robots, in *Proc. 1991 American Nuclear Society Meeting on Robotics and Remote Systems*, Albuquerque, NM, 1991, pp. 299–310.
35. Walker, I. D. and Marcus, S. I., Subtask performance by redundancy resolution for redundant robot manipulators, *IEEE J. Robot. Automat.* **4**(3), 350–354 (1988).
36. Walther, J. S., A unified algorithm for elementary functions, *AFIPS Spring Joint Computer Conf.*, 1971, pp. 379–385.
37. Wang, Y. and Butner, S. E., A new architecture for robot control, in *Proc. 1987 IEEE Conf. Robotics and Automation*, Raleigh, NC, 1987, pp. 664–670.
38. Whitney, D. E., Resolved motion rate control of manipulators and human prostheses, *IEEE Trans. Man Machine Systems* **MMS-10**(2), 47–53 (1969).
39. Yeung, T. B. and Lee, C. S. G., Efficient parallel algorithms and VLSI architectures for manipulator Jacobian calculation, *IEEE Trans. Systems Man Cybernet.* **19**(5), 1154–1166 (1989).