A Parallel Algorithm and Architecture for the Control of Kinematically Redundant Manipulators

Anthony A. Maciejewski, *Member, IEEE,* and J. Michael Reagin, *Member, IEEE*

Abstract— **Kinematically redundant manipulators are inherently capable of more dextrous manipulation due to their additional degrees of freedom. To achieve this dexterity, however, one must be able to efficiently calculate the most desirable configuration from the infinite number of possible configurations that satisfy the end-effector constraint. It has been previously shown that the singular value decomposition (SVD) plays a crucial role in doing such calculations. In this work, a parallel algorithm for calculating the SVD is incorporated into a computational scheme for solving the equations of motion for kinematically redundant systems. This algorithm, which generalizes the damped least squares formulation to include solutions that utilize null-space projections and task prioritization as well as augmented or extended Jacobians, is then implemented on a simple linear array of processing elements. By taking advantage of the error bounds on the perturbation of the SVD, it is shown that an array of only four AT&T DSP chips can result in control cycle times of less than 3 ms for a seven degree-of-freedom manipulator.**

I. Introduction

THE VAST MAJORITY of efforts to utilize redundancy in robotic manipulators have been focused on the resolution of redundancy at the kinematic level. The kinematics of manipulators is frequently represented by

$$\dot{\mathbf{x}} = J\dot{\theta} \tag{1}$$

where $\dot{\mathbf{x}}$ is an m-dimensional vector specifying the end-effector velocity, $\dot{\theta}$ is an n-dimensional vector denoting the joint velocities, and J is the m by n manipulator Jacobian matrix. For redundant manipulators $n > m$ so that the general solution to (1) is typically presented in the form

$$\dot{\theta} = J^{+}\dot{\mathbf{x}} + (I - J^{+}J)\mathbf{z} \tag{2}$$

where $^{+}$ denotes the pseudoinverse and $(I - J^{+}J)\mathbf{z}$ is the projection of an arbitrary vector $\mathbf{z}$ in $\dot{\theta}$ space onto the null space of J. The second term in (2) is the homogeneous solution to (1) since it results in no end-effector velocity and will be denoted here by $\dot{\theta}_H$. This homogeneous solution is frequently used to optimize some secondary criterion under the constraint of the specified end-effector velocity by choosing $\mathbf{z}$ to be the gradient of some function $g(\theta)$ [20]. Some of the secondary criteria that have been applied include joint range availability [19], singularity avoidance [26], [39], various measures of dexterity [5], [8], [11], [18], [31], [40], [41], and obstacle avoidance [23], [39]. The homogeneous solution can also be used to optimize secondary criteria defined in Cartesian space, either to impose a priority to the manipulation variables [28] or to avoid obstacles [23], by using

$$\mathbf{z} = [J_S(I - J^{+}J)]^{+}(\dot{\mathbf{x}}_S - J_S J^{+}\dot{\mathbf{x}}) \tag{3}$$

where the subscript S refers to the secondary criterion. The overall solution is then given by substituting (3) into (2) to obtain

$$\dot{\theta} = J^{+}\dot{\mathbf{x}} + [J_S(I - J^{+}J)]^{+}(\dot{\mathbf{x}}_S - J_S J^{+}\dot{\mathbf{x}}) \tag{4}$$

which has been simplified by taking advantage of the fact that the projection operator is Hermetian and idempotent [23]. An analogous expression at the acceleration level allows the local minimization of joint torque [17].

An alternative to the formulations of (2) and (4) that also utilizes the available redundancy is to include additional kinematic constraints to the original problem described by (1) [35]. If the vector $\dot{\mathbf{x}}$ is augmented with $n - m$ additional kinematic constraints then the resulting Jacobian will be square and traditional inverses can be applied when it is nonsingular [12], [33], [34]. Baillieul [4] has shown that the secondary criterion $g(\theta)$ can be optimized by including the constraint that the gradient of this function be orthogonal to the null space of J. In both of these cases, the extra constraints will introduce algorithmic singularities [4] in addition to the kinematic singularities of the original manipulator.

The issue of dealing with singularities has attracted a great deal of attention [1], [3], [11], [13], [26]. One effective method of dealing with singularities, be they kinematic or algorithmic, is to use the damped least squares formulation that was independently proposed in [27] and [37], and extended in [24]. The damped least squares solution of an equation such as (1) will be denoted by $\dot{\theta}^{(\lambda)}$ and is defined as the solution that minimizes the quantity

$$\|\dot{\mathbf{x}} - J\dot{\theta}\|^2 + \lambda^2\|\dot{\theta}\|^2 \tag{5}$$

where λ is a weighting factor, sometimes referred to as the damping factor, which is used to set the relative importance of satisfying (1) versus the norm of that solution. It is easy to show that the damped least squares solution is a generalization of the pseudoinverse solution since it can be obtained by setting $\lambda = 0$. Therefore, in the remainder of this work the first

Manuscript received October 9, 1992; revised August 17, 1993. This work was supported in part by Sandia National Laboratories under Contract 18-4379B and in part by the NEC Corporation, the TRW Foundation, and General Motors. This paper was presented in part at The 1992 IEEE International Conference on Robotics and Automation, Nice, France, May 12–14, 1992.

A. A. Maciejewski is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907 USA.

J. M. Reagin is with the Inland Fisher Guide Division of General Motors, Anderson, IN USA.

IEEE Log Number 9401089.

term in (2) or (4) will always be calculated as a damped least squares solution. Since the addition of the homogeneous term will always increase the solution norm, it only makes sense to add this term when the joint norm is below its physical constraint thus implying that the damping factor is zero and the pseudoinverse has been calculated. The advantage of using this more general technique is that the same algorithm and architecture can be used for all types of formulations whether they are described by (2), (4), or by an augmented or extended Jacobian.

The remainder of this paper is organized as follows: Section II reviews the properties of the singular value decomposition (SVD) that are relevant to the control of redundant manipulators and presents an algorithm for computing the SVD based exclusively on Givens rotations. Section III discusses the perturbation bounds on the SVD that allow one to avoid the iterative nature of typical implementations. Section IV illustrates how this SVD algorithm can be parallelized and incorporated into a parallel evaluation of solutions described by (2) or (4). Section V then discusses an actual implementation of the complete algorithm on an AT&T pixel machine and presents timing data. The conclusions of this work are then presented in the final section.

II. Review of the SVD

The SVD of the Jacobian is defined as the matrix factorization

$$J = UDV^T \tag{6}$$

where U is an m by m orthogonal matrix of the output singular vectors, V is an n by n orthogonal matrix of the input singular vectors, and D is a diagonal matrix of the form

$$D_{m\times n} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \ddots & \vdots & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 & \vdots & & \vdots \\ 0 & \cdots & 0 & \sigma_m & 0 & \cdots & 0 \end{bmatrix} \tag{7}$$

where the σ_i are the singular values which are typically ordered from largest to smallest. The SVD plays a central role with regard to evaluating the dexterity of manipulator configurations since most local dexterity measures such as manipulability [40], [41], isotropy [2], [31], proximity to singularities [18], fault tolerance [22], compatibility [8], and others [5], [11] are simple functions of the SVD. An efficient implementation of the SVD is also critical for the calculation of (2), (4), and $\dot{\theta}^{(\lambda)}$ since they are also easily described in terms of the SVD. In particular, if (6) is written as

$$J = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{8}$$

where u_i and v_i are the ith columns of U and V respectively, and r is the rank of J, then the pseudoinverse can be formed by using

$$J^+ = \sum_{i=1}^{r} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T \tag{9}$$

and the projection onto the null space by using

$$(I - J^+J) = \sum_{i=r+1}^{n} \mathbf{v}_i \mathbf{v}_i^T. \tag{10}$$

Likewise, the determination of the damped least squares solution which is simply a generalization of (9) is easily obtained by using

$$J^{(\lambda)} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \mathbf{v}_i \mathbf{u}_i^T. \tag{11}$$

Thus an efficient implementation for generating the SVD of the Jacobian plays a critical role in the control of redundant manipulators. While the arguably best general algorithm for calculating the SVD is the Golub–Reinsch algorithm [14], for this application one would like to use an algorithm that is more amenable to parallelization [21]. Thus the algorithm used here is based on Givens rotations [16], [25], [29] which are orthogonal transformations of the form

$$Q = \begin{bmatrix} 1 & & & & & & & & & & & \\ & \ddots & & & & & & & & & & \\ & & 1 & & & & & & & & & \\ & & & \cos(\phi) & & & & -\sin(\phi) & & & & i \\ & & & & 1 & & & & & & & \\ & & & & & \ddots & & & & & & \\ & & & & & & 1 & & & & & \\ & & & \sin(\phi) & & & & \cos(\phi) & & & & j \\ & & & & & & & & 1 & & & \\ & & & & & & & & & \ddots & & \\ & & & & & & & & & & 1 & \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix}$$
$$\begin{matrix} & & & i & & & & j & & & & \end{matrix} \tag{12}$$

where all other elements not shown are zero. This transformation can be geometrically interpreted as a plane rotation of ϕ in the i-j plane. Successive Givens rotations are used to generate the orthogonal matrix V that will result in

$$JV = B \tag{13}$$

where the columns of B are orthogonal. A matrix with orthogonal columns can be written as the product of the orthogonal matrix U and the diagonal matrix D

$$B = UD \tag{14}$$

by letting the columns of U be equal to normalized versions of the columns of B

$$\mathbf{u}_i = \frac{\mathbf{b}_i}{\|\mathbf{b}_i\|} \tag{15}$$

and defining the diagonal elements of D to be equal to the norm of the columns of B

$$\sigma_i = \|\mathbf{b}_i\| \tag{16}$$

thus resulting in the SVD of J.

The critical step in the above procedure is determining the orthogonal matrix V which will orthogonalize the columns of J. This matrix is formed as a product of Givens rotations, each

of which is designed to orthogonalize two columns. Considering the current ith and jth columns of J, multiplication by a Givens rotation results in the new columns, $\mathbf{j}_i'$ and $\mathbf{j}_j'$ given by

$$\mathbf{j}_i' = \mathbf{j}_i \cos(\phi) + \mathbf{j}_j \sin(\phi) \tag{17}$$

$$\mathbf{j}_j' = \mathbf{j}_j \cos(\phi) - \mathbf{j}_i \sin(\phi). \tag{18}$$

The constraint that these columns be orthogonal results in

$$\mathbf{j}_i'^T \mathbf{j}_j' = 0 = \mathbf{j}_i^T \mathbf{j}_j (\cos^2(\phi) - \sin^2(\phi)) + (\mathbf{j}_j^T \mathbf{j}_j - \mathbf{j}_i^T \mathbf{j}_i) \sin(\phi) \cos(\phi). \tag{19}$$

The terms in the Givens rotation matrix to achieve orthogonality can be computed by using the formulas given in [30] which are based on the quantities

$$p = \mathbf{j}_i^T \mathbf{j}_j \tag{20}$$

$$q = \mathbf{j}_i^T \mathbf{j}_i - \mathbf{j}_j^T \mathbf{j}_j \tag{21}$$

$$v = \sqrt{4p^2 + q^2} \tag{22}$$

so that for $q \geq 0$

$$\cos(\phi) = \sqrt{\frac{v+q}{2v}} \quad \text{and} \quad \sin(\phi) = \frac{p}{v \cos(\phi)} \tag{23}$$

and for $q < 0$

$$\sin(\phi) = \operatorname{sgn}(p) \sqrt{\frac{v-q}{2v}} \quad \text{and} \quad \cos(\phi) = \frac{p}{v \sin(\phi)} \tag{24}$$

where

$$\operatorname{sgn}(p) = \begin{cases} 1 & \text{if } p \geq 0 \\ -1 & \text{if } p < 0 \end{cases}. \tag{25}$$

The two sets of formulas are given so that ill-conditioned equations resulting from the subtraction of nearly equal numbers can always be avoided.

If the Givens rotation to orthogonalize columns i and j is denoted by V_{ij} then the matrix V can be computed as the product of a set of $n(n-1)/2$ rotations, referred to as a sweep [15], so that

$$V = \prod_{l=1}^{\#\text{ sweeps}} \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} V_{ij} \right). \tag{26}$$

While the number of sweeps required to orthogonalize the columns of J is not generally known *a priori*, the following section illustrates that by using information from the SVD of the previous J one can typically obtain V in a single sweep.

It is instructive to perform a comparison between the computational requirements of this proposed algorithm with other standard efficient matrix techniques for solving equations in the form of (2) or (4). Solutions to (2) and (4), as well as damped least squares solutions, are all special cases of the general class of least squares problems with linear equality constraints. Thus all of them can be solved by traditional least squares algorithms applied to an appropriately weighted augmented Jacobian for which the dimension m is now greater than n. The computational requirements for a number of such standard techniques are presented in the Table I with a detailed description of the algorithms available in [15].

TABLE I
COMPARISON OF ALGORITHM OPERATION COUNTS

Algorithm	Asymptotic FLOP count
Normal equations (Cholesky)	$mn^2/2 + n^3/6$
Householder orthogonalization	$mn^2 - n^3/3$
Modified Gram-Schmidt	mn^2
Givens orthogonalization	$2mn^2 - 2n^3/3$
Golub-Reinsch SVD	$2mn^2 + \approx 4n^3$

As can be seen from the Table I, the most computationally efficient technique is to apply the Cholesky algorithm to the normal equations. However, this technique can not be recommended for all situations since the conditioning of the original problem is squared while forming the normal equations. The orthogonalization methods are all improvements in terms of numerical stability, however, they all require additional computational expense. The superiority of the Householder orthogonalization algorithm in terms of computational requirements is somewhat misleading since the Givens orthogonalization algorithm, like the proposed SVD algorithm based on Givens rotations, is much more amenable to parallel implementations. In terms of numerical reliability when dealing with ill-conditioned equations, the SVD is unequivocally superior to all other techniques, however, this reliability is obtained with an increase in the computational expense. The operation counts for the Golub-Reinsch algorithm are only approximate due to the iterative nature of the algorithm and are for the case when only the solution to the least squares problem is desired and not the actual decomposition. By comparison, the SVD algorithm presented here requires $3n$ FLOP's to determine the rotation angle to orthogonalize two columns and then another $8n$ FLOP's to apply these rotations to both J and V. Thus if one were to perform a single sweep serially on a single processor the resulting FLOP count would be on the order of $11n^3/2$, which would not compare favorably with the above techniques. However, in Section IV it will be shown how the Givens rotations can be done in parallel using $n/2$ processors so that the operation count becomes $11n^2$, which is superior to any serial implementation of the algorithms shown in Table I. It is important to emphasize, however, that the primary advantage of this technique is not necessarily computational efficiency but rather the improved performance of kinematically redundant manipulators that is possible when using the insight gleaned from the complete SVD of the Jacobian.

III. PERTURBATION BOUNDS ON THE SVD

If one considers θ_k to be the current configuration of the manipulator, then the SVD of J from the previous computation cycle time is known and is given by

$$J(\theta_{k-1}) = U(\theta_{k-1}) D(\theta_{k-1}) V^T(\theta_{k-1}). \tag{27}$$

If one considers the current manipulator Jacobian to be a perturbation of the previous Jacobian

$$J(\theta_k) = J(\theta_{k-1}) + \Delta J(\theta_{k-1}) \tag{28}$$

then the matrix $J(\theta_k)V(\theta_{k-1})$ will have nearly orthogonal columns provided that $\Delta J(\theta_{k-1})$ is small compared to $J(\theta_{k-1})$ [9], [38]. Therefore, in this work (26) is calculated using

$$V(\theta_k) = V(\theta_{k-1})\left(\prod_{i=1}^{n-1}\prod_{j=i+1}^{n} V_{ij}\right) \tag{29}$$

where only a single sweep is performed to update the value of $V(\theta_{k-1})$. While restricting the number of sweeps *a priori* has a number of advantages, including the elimination of convergence tests, it is important to consider the robustness of such an approximation. A qualitative discussion of the types of configurations that will introduce errors into this approximation is found in [25], however, the quantitative effect of $\Delta J(\theta_{k-1})$ has not been previously presented.

It is instructive to first consider the accuracy of a computed SVD based on a single sweep without using any prior information. Fig. 1 presents data on the accuracy to which the calculated SVD approximates the desired Jacobian when the algorithm is initialized with $V(\theta_k)$ equal to the identity matrix and then halted after a single sweep. Each point on this graph represents the average error over an entire randomly selected joint space trajectory for the seven degree-of-freedom CESARM manipulator [10]. The trajectories are traversed at different speeds so that with a constant computation cycle time the change between two consecutive configurations in a trajectory, i.e. $\theta_k - \theta_{k-1}$ will vary in magnitude. The range of this magnitude is varied from 0.1 to 1.9 rad at a resolution of 0.1 rad with 300 random trajectories calculated for each value of $||\theta_k - \theta_{k-1}||$. From this figure it is clear that a single sweep of the proposed algorithm, even without using prior information, provides a reasonable approximation to the SVD of J. The average errors in this approximation typically lie in the range of 2–4% with none being greater than 7%. Since the algorithm is always initialized with the identity matrix there is no variation with respect to the magnitude of the change between successive configurations in the trajectory, as would be expected.

Fig. 2 presents data from a simulation identical to that described for Fig. 1, except that now the algorithm initializes $V(\theta_k)$ to $V(\theta_{k-1})$ before performing the single sweep. This approach clearly provides greater accuracy in the computed SVD even when the distance between successive configurations is as high as 1 rad. For configurations that are separated by more than 1 rad, the accuracies are similar to those in Fig. 1 since there is little correlation between $V(\theta_k)$ and $V(\theta_{k-1})$ so that $V(\theta_{k-1})$ and the identity matrix are both effectively random initializations. In most practical implementations it is easy to show that the change in manipulator configurations between successive sampling intervals will typically be below 0.1 rad so that the accuracy of the SVD calculation will be well within 1%. In particular, in Section V it will be shown that the computation cycle time for this algorithm is below 3 ms so that

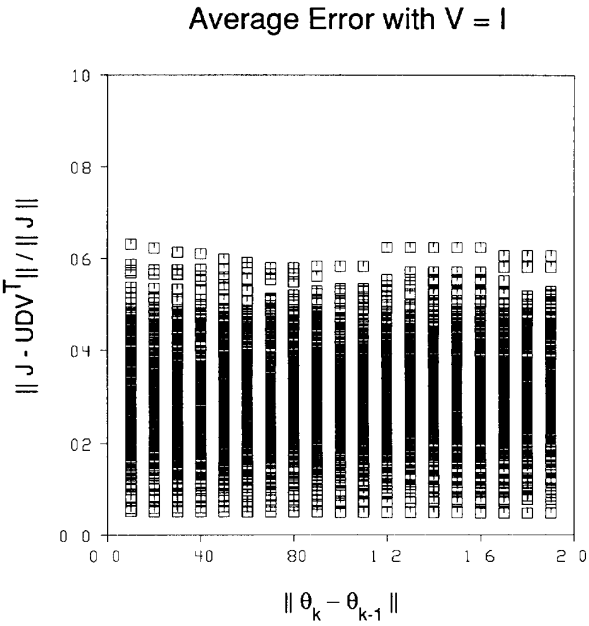


Fig. 1. This graph shows the percentage of error in the calculated SVD of the Jacobian if the proposed algorithm is limited to a single sweep with the value of V initialized to the identity matrix. Each data point on the graph represents the average error over an entire randomly selected joint-space trajectory of the seven degree-of-freedom CESARM manipulator [10]. Data is shown for joint speeds that result in successive sampled joint configurations ranging from 0.1 to 1.9 rad. There are 300 trajectories calculated for each value of joint speed.

a change of 0.1 rad corresponds to an average angular velocity of over 33 rad/s. It should be noted that a significant portion of the 300 random trajectories that are plotted in Fig. 2 naturally contain configurations in which the resulting Jacobian is singular and that these configurations pose no special concerns for the proposed algorithm. The configurations which result in the largest errors for a given perturbation are those in which three or more singular values are clustered together resulting in significant interaction between successive Givens rotations. A more detailed discussion on the characteristics of manipulator configurations that result in the largest errors is available in [25].

IV. A Parallel Architecture and Algorithm

This section discusses a simple parallel architecture and the implementation of the above algorithm on that architecture so that solutions in the form of (2), (4), or $\dot{\theta}^{(\lambda)}$ can be calculated in real time. The architecture to solve these kinematic equations of motion consists of a host processor and a linear array of $n/2$ processing elements (PE's). This section and the one that follows will consider an array that contains four PE's and which is suitable for manipulators with up to eight degrees of freedom as an example, however, the results are completely general. Each PE in the linear array can exchange data with the PE on its right or left as well as with the host processor. It should be mentioned that it is possible to use one of the processors in the linear array to provide the functions of the host processor, however, for the purposes of

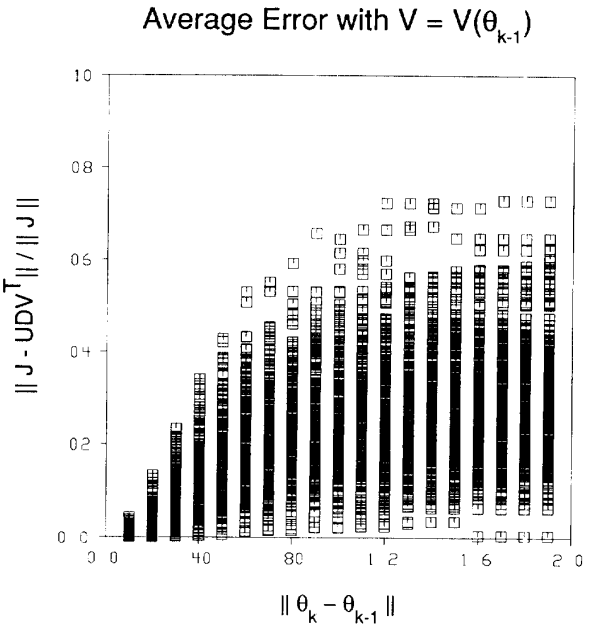


Fig. 2. This graph shows the percentage of error in the calculated SVD of the Jacobian if the proposed algorithm is limited to a single sweep with the value of V initialized to the value of V from the previous configuration. Each data point on the graph represents the average error over an entire randomly selected joint-space trajectory of the seven degree-of-freedom CESARM manipulator [10]. Data is shown for joint speeds that result in successive sampled joint configurations ranging from 0.1 to 1.9 rad. There are 300 trajectories calculated for each value of joint speed.

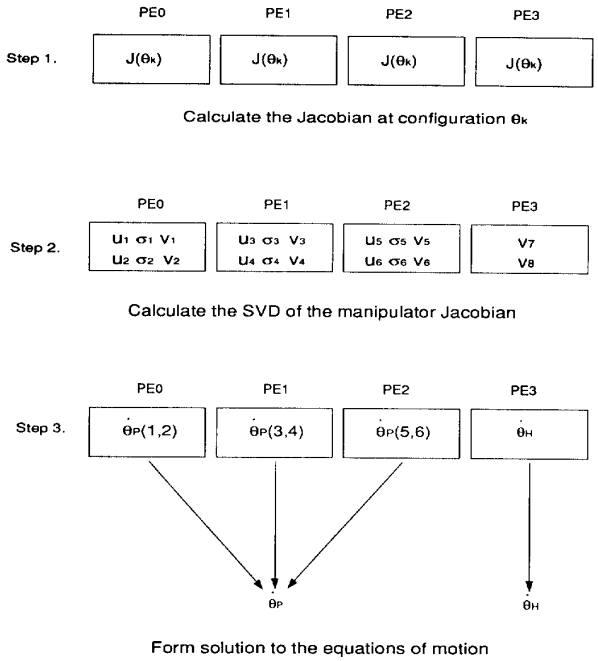


Fig. 3. The three steps required to solve the equations of motion for kinematically redundant manipulators. The final solution is composed of a component that satisfies the specified end-effector velocity, denoted $\dot{\theta}_P$, and a component that represents the redundant degrees of freedom, denoted $\dot{\theta}_H$.

illustration it will be described as a separate processing entity. This architecture is quite similar to that proposed in [36] for the control of redundant manipulators, however, it does not require the specialized VLSI implementation of a CORDIC SVD processor [7].

Obtaining solutions in the form of (2) or (4) involves three distinct steps as shown in Fig. 3. The first step is the calculation of the end-effector Jacobian, J, which is computed locally on each of the PE's with the host sending only the current manipulator configuration θ_k. The second step involves the calculation of the SVD of the Jacobian and the third step involves forming either (2) or (4) using the SVD to form the damped least squares inverse of J, the projection operator $(I - J^+J)$, and for (4), the damped least squares inverse of $[J_S(I - J^+J)]$. The parallel execution of these steps will now be considered in detail.

When the current end-effector Jacobian $J(\theta_k)$ is being calculated in the four PE's, the SVD of the previous Jacobian is available in these PE's with each of the PE's containing two singular values and the input and output singular vectors associated with these singular values. The first step in calculating the SVD of the current Jacobian, identified as step 2.1 in Fig. 4, is to multiply the current Jacobian $J(\theta_k)$ by $V(\theta_{k-1})$, the V matrix associated with the SVD of the previous Jacobian. Assuming that the magnitude of the joint velocities are physically realistic, $J(\theta_k)V(\theta_{k-1})$ will have nearly orthogonal columns due to the perturbation bounds discussed in the previous section.

The second step in calculating the SVD is the performance of a single sweep of Given's rotations with column shift ordering as described in [32]. This particular column shift ordering allows the columns of B and V to be exchanged between PE's using a series of shifts where all PE's either shift their left columns to the right or their right columns to the left. The advantage of using this type of shifting is that it simplifies the control and only requires a single bi-directional data path between PE's. The disadvantage, however, is that a PE is idle during every other step during the sweep. If two data paths are provided between PE's then the column shift ordering shown in Fig. 5 can be used to eliminate the idle PE [6]. In this work the simpler shift ordering is used so that there are eight steps, denoted 2.2.1 to 2.2.8 in Fig. 4, required to perform the single sweep. In this figure, $\mathbf{v}_i$ denotes the columns of the V matrix which are a product of all the Givens rotations performed up to that point and $\mathbf{b}_i$ denotes the columns of the current B matrix which is the Jacobian $J(\theta_k)$ multiplied by the matrix V. The sweep begins at step 2.2.1 with each PE performing a Given's rotation on its two columns of B. As discussed before, the angle of the plane rotation is selected to make these two columns orthogonal. The PE's then shift their left column of B and V to the processor on the right. Note that the right-most processor now has three columns of B and V. Now at step 2.2.2 each PE, with the exception of the left-most one, performs a Given's rotation on its two columns. Each processor then shifts its right column of B and V to the PE on its left. This process of left and right shifts continues until step 2.2.8 when eight levels of Given's rotations, i.e., a single

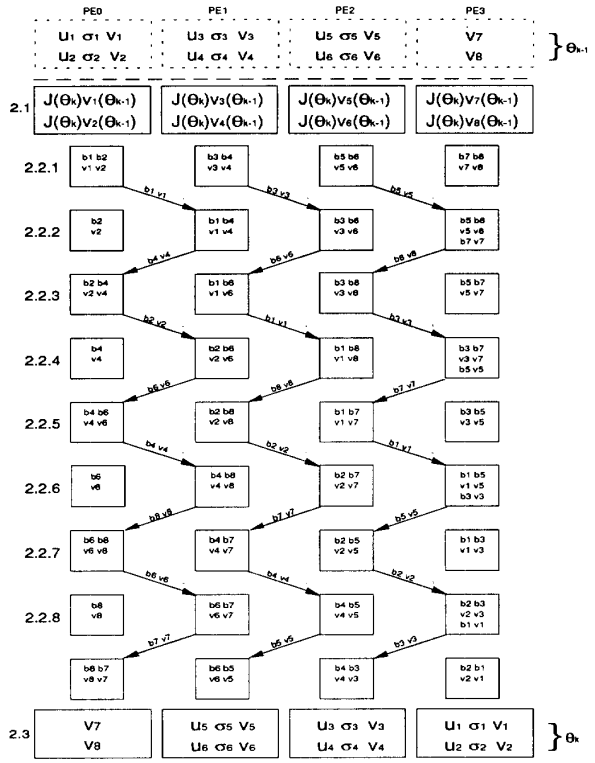


Fig. 4. Step 2 of the overall algorithm consists of calculating the SVD of the Jacobian. This is done by first multiplying the Jacobian by the previous matrix V in step 2.1, then performing a single sweep of Givens rotations in steps 2.2.1–2.2.8, and finally normalizing the resulting columns of the matrix B to determine the matrix U and the singular values.

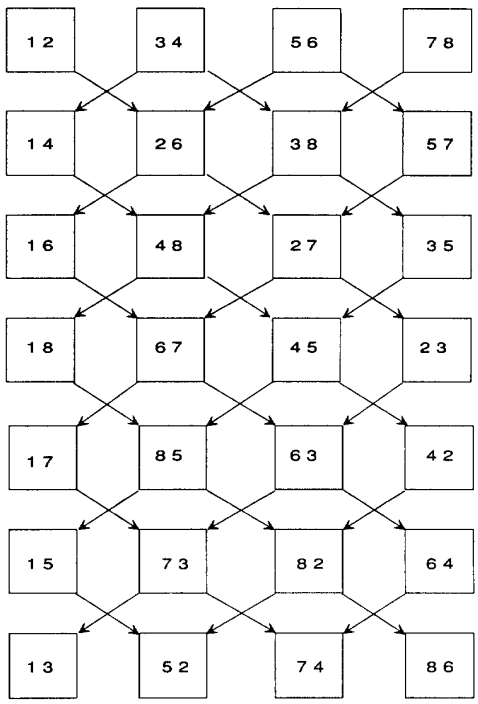


Fig. 5. An alternate method of doing the column shifts to perform a single sweep of Givens rotations that does not result in any idle PE's. This technique requires two data paths between PE's for simultaneous transfers.

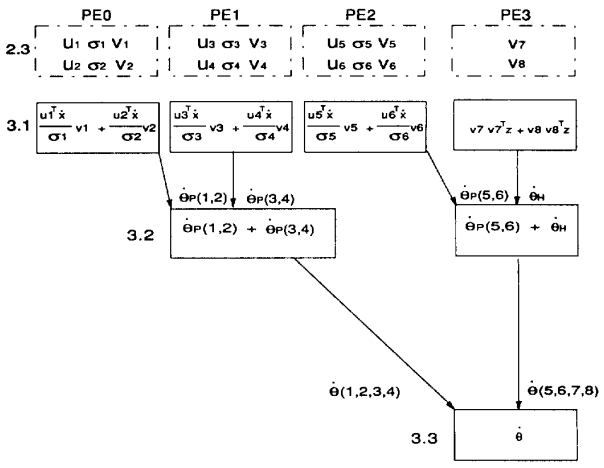


Fig. 6. An illustration of how the SVD of J can be used to calculate solutions in the form of $\dot{\theta} = J^{+}\dot{\mathbf{x}} + (I - J^{+}J)\mathbf{z}$.

sweep, have been performed. The last step in determining the SVD involves calculating the matrix $U(\theta_k)$ using (15) and the singular values using (16).

Once the SVD of J has been computed, the third step, i.e., calculation of a solution in the form of (2) or (4), is greatly simplified. The first term of either equation, denoted by $\dot{\theta}_P$, is considered as a damped least squares solution given by

$$\dot{\theta}_P = \dot{\theta}^{(\lambda)} = J^{(\lambda)}\dot{\mathbf{x}} \tag{30}$$

which is easily computed using the SVD of J by applying (11). Note once again that this provides the pseudoinverse solution if $\lambda = 0$. The second term of (2) is also easily computed once the SVD is available by using (10). Fig. 6 illustrates the calculation of (2) on the parallel architecture for the case where $m = 6$ and $n = 8$. At the end of step 2.3, each PE has two columns of the SVD of J. The PE's which have columns 1 to 6 of U, D, and V compute their contribution to $\dot{\theta}_P$ in step 3.1 as

$$\dot{\theta}_P(i, i+1) = \frac{\sigma_i}{\sigma_i^2 + \lambda^2}\mathbf{v}_i\mathbf{u}_i^T\dot{\mathbf{x}} + \frac{\sigma_{i+1}}{\sigma_{i+1}^2 + \lambda^2}\mathbf{v}_{i+1}\mathbf{u}_{i+1}^T\dot{\mathbf{x}} \tag{31}$$

while the PE with columns 7 and 8 computes the term

$$\dot{\theta}_H = (I - J^{+}J)\mathbf{z} = \mathbf{v}_7\mathbf{v}_7^T\mathbf{z} + \mathbf{v}_8\mathbf{v}_8^T\mathbf{z} \tag{32}$$

which is the second term of (2). If the value of the damping factor was zero then the intermediate terms are summed to form $\dot{\theta}$, as defined by (2), in steps 3.2 and 3.3.

The computation of (4) is more difficult than (2) due to the calculation of the projection

$$A^{+} = [J_S(I - J^{+}J)]^{+}. \tag{33}$$

However, it can be shown that the calculation of the SVD of A^{+} is not as expensive as it might first seem. It has been

previously shown [23] that

$$(I - J^+J)[J_S(I - J^+J)]^+ = [J_S(I - J^+J)]^+ \quad (34)$$

so that the range of A^+ is known to be a subset of the null space of J. Therefore, to compute its SVD the Givens rotations can be restricted to the $(n-r)$-dimensional space described by the singular vectors v_{r+1} to v_n rather than the full n-dimensional space. As an example, consider the case of a seven degree-of-freedom manipulator with a one-dimensional null space so that the SVD of A is given by

$$A = \sigma_{A_1}\mathbf{u}_{A_1}\mathbf{v}_7^T \quad (35)$$

where $\mathbf{v}_7$ describes the null space of J. It is easy to show that the only possibly non-zero singular value of A can be computed by using

$$\sigma_{A_1} = \|J_S\mathbf{v}_7\|. \quad (36)$$

If $\sigma_{A_1} \neq 0$ then the singular vector $\mathbf{u}_{A_1}$ associated with this singular value is computed using

$$\mathbf{u}_{A_1} = \frac{J_S\mathbf{v}_7}{\sigma_{A_1}}. \quad (37)$$

Similarly for a two-dimensional null space, the SVD of A, given by

$$A = \sigma_{A_1}\mathbf{u}_{A_1}\mathbf{v}_{A_1}^T + \sigma_{A_2}\mathbf{u}_{A_2}\mathbf{v}_{A_2}^T \quad (38)$$

can be computed using a greatly simplified procedure. First, the matrix J_S is multiplied by the singular vectors that form the null space of J

$$\mathbf{b}_{A_1} = J_S\mathbf{v}_7 \quad (39)$$

$$\mathbf{b}_{A_2} = J_S\mathbf{v}_8. \quad (40)$$

which results in a basis for the range of A^+. Next, a single Givens rotations is performed to orthogonalize these columns and determine the two input singular vectors

$$\mathbf{b}'_{A_1} = \mathbf{b}_{A_1}\cos(\phi) + \mathbf{b}_{A_2}\sin(\phi) \quad (41)$$

$$\mathbf{b}'_{A_2} = \mathbf{b}_{A_2}\cos(\phi) - \mathbf{b}_{A_1}\sin(\phi) \quad (42)$$

$$\mathbf{v}_{A_1} = \mathbf{v}_7\cos(\phi) + \mathbf{v}_8\sin(\phi) \quad (43)$$

$$\mathbf{v}_{A_2} = \mathbf{v}_8\cos(\phi) - \mathbf{v}_7\sin(\phi). \quad (44)$$

The singular values and output singular vectors of A are then computed using (15) and (16) on $\mathbf{b}'_{A_1}$ and $\mathbf{b}'_{A_2}$. For higher degrees of redundancy the vectors $\mathbf{b}_{A_1}$ through $\mathbf{b}_{A_{n-r}}$ can be orthogonalized by using sweeps of Givens rotation that are restricted to this $n-r$ dimensional subspace, a task for which this architecture has been optimized.

Fig. 7 illustrates the computations required to obtain a solution in the form of (4) on the parallel architecture for the case where $m = 6$ and $n = 8$. As in the computation of (2), the PE's with columns 1 to 6 compute their contribution to $\dot{\theta}_P$ using (11) in step 3.1. The PE with columns 7 and 8 is responsible for computing the SVD of $[J_S(I - J^+J)]^+$ using the method described above. In step 3.2 the results from the three PE's are combined to form $\dot{\theta}_P$ which is then distributed

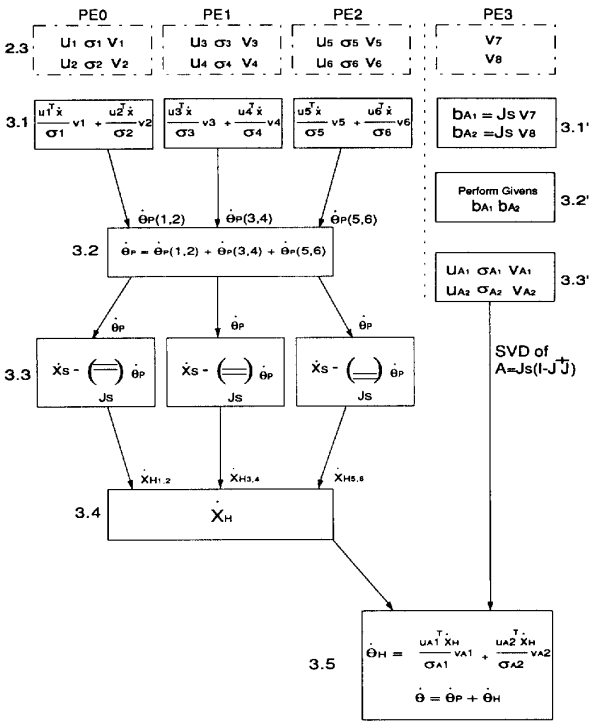


Fig. 7. An illustration of how the SVD of J can be used to calculate solutions in the form of $\dot{\theta} = J^+\dot{\mathbf{x}} + [J_S(I - J^+J)]^+(\dot{\mathbf{x}}_S - J_SJ^+\dot{\mathbf{x}})$.

back to each of the three PE's in step 3.3 so they can each compute two elements of the intermediate term

$$\dot{\mathbf{x}}_H = \dot{\mathbf{x}}_S - J_S\dot{\theta}_P \quad (45)$$

which are then combined in step 3.4. Once $\dot{\mathbf{x}}_H$ and A^+ are available the second term of (4) is formed using

$$\dot{\theta}_H = \frac{\sigma_{A_1}}{\sigma_{A_1}^2 + \lambda^2}\mathbf{v}_{A_1}\mathbf{u}_{A_1}^T\dot{\mathbf{x}}_H + \frac{\sigma_{A_2}}{\sigma_{A_2}^2 + \lambda^2}\mathbf{v}_{A_2}\mathbf{u}_{A_2}^T\dot{\mathbf{x}}_H. \quad (46)$$

The value sent to the joint controller is then obtained by adding $\dot{\theta}_P$ and $\dot{\theta}_H$.

Note that the benefit of performing the calculation of $\dot{\mathbf{x}}_H$ in parallel for (4) must be weighed against the amount of communication overhead required. Depending on the ratio of the computation time to communication time, it may be desirable to perform the calculation of $\dot{\mathbf{x}}_H$ serially. This decision will be affected by the communication rate and the size of the $n-r$ dimensional subspace of J. For larger values of $n-r$, the time associated with the computation of the SVD of A will be much larger than the computation time of $\dot{\mathbf{x}}_H$.

V. Implementation

The algorithm described in the previous section was implemented on a linear array of eight DSP32 processors that were part of an AT&T Pixel machine with a Sun workstation serving as the host processor. This configuration permits timing evaluations for manipulator systems with up to sixteen degrees of freedom. For the example discussed above only four PE's were

TABLE II
TIMING STATISTICS FOR CALCULATING SOLUTIONS IN THE FORM OF (2) AND (4). (PARALLEL TIMES REPRESENT FOUR DSP32 PE'S)

$\dot{\theta} = J^{+}b\dot{x} + (I - J^{+}J)z$

	Serial	Parallel	Speedup
T_{SVD}	6.2 ms	1.7 ms + 8 T_{SHIFT}	3.65
T_{SOLVE}	0.5 ms	0.2 ms	3.31
Total	6.7 ms + $T_{JACOBIAN}$	1.9 ms + $T_{JACOBIAN}$ + 8 T_{SHIFT}	3.61

$\dot{\theta} = J^{+}b\dot{x} + [J_S(I - J^{+}J)]^{+}(\dot{x}_S - J_S J^{+}\dot{x})$

	Serial	Parallel	Speedup
T_{SVD}	6.2 ms	1.7 ms + 8 T_{SHIFT}	3.65
T_{SOLVE}	1.4 ms	0.7 ms	1.89
Total	7.6 ms + $T_{JACOBIAN}$	2.4 ms + $T_{JACOBIAN}$ + 8 T_{SHIFT}	3.17

required. The DSP32 processors are rated at 5MFLOPS with a serial transfer rate between processors of 5.25 Mbits/s. This platform was selected due to its software support tools and availability despite the fact that the serial transfer rate between PE's presents a serious drawback. However, the execution time of the algorithm on this architecture can be extrapolated to architectures that support a higher bandwidth communication between PE's. The algorithm was implemented in C and then compiled for the DSP32 PE's. While there was an effort to efficiently utilize register variables to improve performance, the execution times are by no means to be considered optimal.

The time required for the calculation of $\dot{\theta}$, denoted by $T_{\text{CALC\_}\dot{\theta}}$, is given by

$$T_{\text{CALC\_}\dot{\theta}} = T_{\text{JACOBIAN}} + T_{\text{SVD}} + T_{\text{SOLVE}} \qquad (47)$$

where T_{JACOBIAN} is the time to acquire the joint positions and calculate the current Jacobian, T_{SVD} is the time to calculate the SVD of the end-effector Jacobian J, and T_{SOLVE} is the time required to form the solution of (2) or (4) using the SVD of J. Timing information for both serial and parallel implementations of (2) and (4) on the AT&T pixel machine is presented in Table II. Note that the values of T_{JACOBIAN} and T_{SHIFT} are left as variable parameters. The value of T_{JACOBIAN} is not specified in the table since it is manipulator dependent. The equations for calculating the Jacobian of the CESAR manipulator [10], which required approximately 0.38 ms, were implemented in order to provide some comparison with the other execution times. The parameter T_{SHIFT}, which is the time required to shift a column of B and V to an adjacent PE, is also given separately since it becomes negligible when dual-port RAMs are substituted for the serial links between PE's. Table II shows that once the SVD of the manipulator Jacobian has been calculated, the solution to either (2) or (4) can be computed in a relatively short period of time. This is a major motivation for using the SVD in the computation of these equations. Since the calculation of the SVD accounts for the majority of the computation time, approximately 1.7 ms, the speedup ratio (3.65) associated with the SVD has the greatest effect on the overall performance improvement for the calculation of (2) or (4).

TABLE III
BREAKDOWN OF THE TIME REQUIRED TO CALCULATE THE SVD. (PARALLEL TIMES REPRESENT FOUR DSP32 PE'S)

	Step	Serial	Parallel	Speedup
T_{MULT}	2.1	0.88 ms	0.22 ms	4.0
T_{SWEEP}	2.2	4.50 ms	1.29 ms	3.5
$T_{\text{NORM B}}$	2.3	0.88 ms	0.22 ms	4.0

Table III presents more information on the time required to calculate the SVD by analyzing its components which are given by

$$T_{\text{SVD}} = T_{\text{MULT V}} + T_{\text{SWEEP}} + T_{\text{NORM B}} \qquad (48)$$

where $T_{\text{MULT V}}$ denotes the time required to multiply $J(\theta_k)$ by $V(\theta_{k-1})$ on the PE's, T_{SWEEP} is the time to perform a single sweep of Givens rotations, and $T_{\text{NORM B}}$ is the time required to calculate the singular values and columns of U from the columns of B at the end of the sweep. From Table III one can see that there is a factor of four speedup for $T_{\text{MULT V}}$ and $T_{\text{NORM B}}$ and a 3.5 times speedup for T_{SWEEP} illustrating the extremely parallel nature of this particular algorithm. The theoretical maximum speedup of four could be achieved for T_{SWEEP} as well by using the column ordering shown in Fig. 5 thus removing the idle PE in every other step as discussed previously [6].

The computation of (2) in parallel, which is shown in Fig. 6, is 3.6 times faster than the calculation of (2) in serial for the case where $n = 8$. The two components involved in the computation of (2) are T_{SVD} and $T_{SOLVE\ (2)}$. The most important thing to note is the relatively insignificant amount of time required to solve (2) once the SVD of the manipulator Jacobian has been calculated. The overall speedup factor for the component $T_{\text{SOLVE (2)}}$ is 3.3. This is mostly due to step 3.1 of the calculations for $T_{\text{SOLVE (2)}}$ where there is almost a factor of four speedup except that there are two fewer divisions on PE3 since it is calculating $\dot{\theta}_H$ rather than contributing to $\dot{\theta}_P$. In this timing example $\mathbf{z}$ was calculated as the gradient of the squares of the joints angles in order to utilize the redundancy for joint range availability [19].

Fig. 7 shows the computation of (4) on the parallel architecture. In Table IV the time $T_{\text{SOLVE (4)}}$ is broken down into $T_{J_S\ \text{MULT}}$, which is the time required to multiply J_S times $\mathbf{v}_7$ and $\mathbf{v}_8$, T_{GIVEN}, which is the time required to perform the single Givens rotation, $T_{\text{NORM B}}$, which is the time required to find σ_{A1}, σ_{A2}, $\mathbf{u}_{A1}$ and $\mathbf{u}_{A2}$ from $\mathbf{b}'_{A1}$ and $\mathbf{b}'_{A2}$, $T_{\text{CALC\_}\dot{x}_H}$, which is the time required to compute the intermediate term $\dot{\mathbf{x}}_H$, and T_{COMM}, which is the communication time required to transfer intermediate terms between the processing elements in steps 3.1 through 3.5. Table II shows that there is a 3.2 times overall speedup in the calculation of (4) on the parallel architecture and a 1.9 times speedup for $T_{\text{SOLVE (4)}}$. As mentioned before, the speedup associated with T_{SVD} has the largest effect since the computation time associated with the SVD of the Jacobian is a major portion of the total computation time. In Fig. 7, steps 3.1′ to 3.3′, the computation of the SVD of A, are executed in 0.6 milliseconds. Note that for

TABLE IV
BREAKDOWN OF THE TIME REQUIRED TO SOLVE (4) ONCE THE SVD IS AVAILABLE. (PARALLEL TIMES REPRESENT FOUR DSP32 PE'S)

Step		Serial	Parallel	Speedup
3.1′	$T_{J_S \text{ MULT}}$	0.22 ms	0.22 ms	—
3.2′	T_{GIVEN}	0.16 ms	0.16 ms	—
3.3′	$T_{NORM\ B}$	0.22 ms	0.22 ms	—
3.1′–3.3′		0.60 ms	0.60 ms	—
3.1		0.38 ms	0.13 ms	3.0
3.2		0.02 ms	ms	—
3.3		0.22 ms	0.02 ms	3.0
3.1–3.4	$T_{CALC\ b\hat{f}x_H}$	0.62 ms	0.22 ms $+T_{COMM}$	2.8
3.5		0.14 ms	0.14 ms	—
3.1–3.4, 3.1′–3.3′, 3.5	$T_{SOLVE\ (4)}$	1.40 ms	0.74 ms	1.9

the case where $n = 8$ these calculations are performed on a single processor which is the reason that there is no speedup for these individual calculations. However, for manipulators with higher degrees of redundancy step 3.2′ will not be a single Givens rotation, but will instead be several sweeps of Givens rotations. This means that the parallel architecture will be of even greater benefit in the higher order cases since the processors with columns $r + 1$ to n will be performing the sweeps of Givens rotations to compute the SVD of A rather than a single processor. In steps 3.1 through 3.4 the calculation of the intermediate term $\dot{\mathbf{x}}_H$ is performed. For a manipulator with one or two degrees of redundancy, the calculation time for $\dot{\mathbf{x}}_H$ on a single processor is greater than the calculation time for the SVD of A in steps 3.1′ to 3.3′. This is the reason that all three processors are used in steps 3.3 to 3.4 even though it increases the complexity of the software. However, for manipulators with higher degrees of redundancy the time to calculate the SVD of A will be much greater than the calculation time of $\dot{\mathbf{x}}_H$. This means that the calculation time for $\dot{\mathbf{x}}_H$ will be totally overlapped by steps 3.1′ to 3.3′ allowing one to simplify the software.

VI. CONCLUSION

This work has presented a parallel algorithm and architecture for solving the equations of motion for kinematically redundant robotic systems. It has been shown that the various desirable forms of the solutions to these equations can all be efficiently obtained when the SVD of the Jacobian is available. Thus a major portion of the algorithm is devoted to the parallel computation of the SVD. The implementation of this algorithm on an array of AT&T DSP32 processors has illustrated that computation cycle times of less than 3 ms can be obtained even when using the most complicated form of solution. By using currently available processors such as the TI TMS320C4x in the same configuration, control frequencies of well over 1 kHz are feasible.

REFERENCES

[1] E. W. Aboaf and R. P. Paul, "Living with the singularity of robot wrists," in *IEEE Int. Conf. Robotics Autom.*, Raleigh, NC, Mar. 31–Apr. 3, 1987, pp. 1713–1717.

[2] J. Angeles, "The design of isotropic manipulator architectures in the presence of redundancies," *Int. J. Robotics Res.*, vol. 11, no. 3, pp. 196–201, June 1992.

[3] H. Asada and J. A. Cro Granito, "Kinematic and static characterization of wrist joints and their optimal design," in *IEEE Int. Conf. Robotics Autom.*, St. Louis, MO, Mar. 25–28, 1985, pp. 244–250.

[4] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *IEEE Int. Conf. Robotics Autom.*, St. Louis, MO, Mar. 25–28, 1985, pp. 722–728.

[5] J. Baillieul, "A constraint oriented approach to inverse problems for kinematically redundant manipulators," in *IEEE Int. Conf. Robotics Autom.*, Raleigh, NC, Mar. 31–Apr. 3, 1987, pp. 1827–1833.

[6] R. P. Brent, F. T. Luk, and C. Van Loan, "Computation of the singular value decomposition using mesh-connected processors," *J. VLSI and Computer Syst.*, vol. 1, no. 3, pp. 242–271, 1985.

[7] J. R. Cavallaro and F. T. Luk, "CORDIC arithmetic for an SVD processor," *J. Parallel and Distributed Computing*, vol. 5, pp. 271–290, 1988.

[8] S. L. Chiu, "Task compatibility of manipulator postures," *Int. J. Robotics Res.*, vol. 7, no. 5, pp. 13–21, 1988.

[9] C. Davis and W. M. Kahan, "The rotation of eigenvectors by a perturbation," *SIAM J. Numerical Anal.*, vol. 7, no. 1, pp. 1–46, 1970.

[10] R. V. Dubey, J. A. Euler, and S. M. Babcock, "An efficient gradient projection optimization scheme for a seven-degree-of-freedom redundant robot with spherical wrist," in *IEEE Int. Conf. Robotics Autom.*, Philadelphia, PA, Apr. 24–29 1988, pp. 28–36.

[11] R. Dubey and J. Y. S. Luh, "Redundant robot control for higher flexibility," in *IEEE Int. Conf. Robotics Autom.*, Raleigh, NC, Mar. 31–Apr. 3, 1987, pp. 1066–1072.

[12] O. Egeland, "Task-space tracking with redundant manipulators," *IEEE Trans. Robotics Automat.*, vol. 3, no. 5, pp. 471–475, 1987.

[13] O. Egeland, J. R. Sagli, I. Spangelo, and S. Chiaverini, "A damped least-square solution to redundancy resolution," in *IEEE Int. Conf. Robotics Automat.*, Sacramento, CA, Apr. 9–11, 1991, pp. 945–951.

[14] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numer. Math.*, vol. 14, pp. 403–420, 1970.

[15] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: Johns Hopkins University Press, 1983.

[16] M. R. Hestenes, "Inversion of matrices by biorthogonalization and related results," *J. Soc. Ind. Appl. Math.*, vol. 6, no. 1, pp. 51–90, 1958.

[17] M. Hollerbach and K. C. Suh, "Redundancy resolution of manipulators through torque optimization," *IEEE Trans. Robotics Automat.*, vol. 3, no. 4, pp. 308–316, 1987.

[18] C. A. Klein and B. E. Blaho, "Dexterity measures for the design and control of kinematically redundant manipulators," *Int. J. Robotics Res.*, vol. 6, no. 2, pp. 72–83, 1987.

[19] C. A. Klein and C. H. Huang, "Review of pseudoinverse control for use with kinematically redundant manipulators," *IEEE Trans. Syst., Man, and Cyber.*, vol. 13, no. 2, pp. 245–250, 1983.

[20] A. Liégeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Trans. Syst., Man, and Cyber.*, vol. 7, no. 12, pp. 868–871, 1977.

[21] F. T. Luk, "A triangular processor array for computing singular values," *Linear Algebra and Its Applications*, vol. 77, pp. 259–273, 1986.

[22] A. A. Maciejewski, "Fault tolerant properties of kinematically redundant manipulators," in *IEEE Int. Conf. Robotics Automat.*, Cincinnati, OH, May 13–18 1990, pp. 638–642.

[23] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. J. Robotics Res.*, vol. 4, no. 3, pp. 109–117, 1985.

[24] A. A. Maciejewski and C. A. Klein, "Numerical filtering for the operation of robotic manipulators through kinematically singular configurations," *J. Robotic Syst.*, vol. 5, no. 6, pp. 527–552, 1988.

[25] A. A. Maciejewski and C. A. Klein, "The singular value decomposition: Computation and applications to robotics," *Int. J. Robotics Res.*, vol. 8, no. 6, pp. 63–79, 1989.

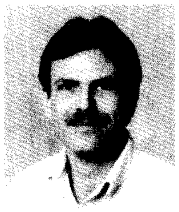
[26] R. V. Mayorga, N. Milano, and A. K. C. Wong, "A fast procedure for manipulator inverse kinematics computation and singularities prevention," *J. Robotic Syst.*, vol. 9, no. 8, 1992.

[27] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *ASME J. Dynamic Syst., Measurement, and Control*, vol. 108, no. 3, pp. 163–171, 1986.

[28] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *Int. J. Robotics Res.*, vol. 6, no. 2, pp. 3–15, 1987.

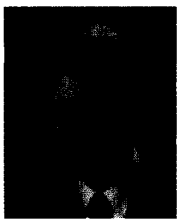
[29] J. C. Nash, "A one-sided transformation method for the singular value decomposition and algebraic eigenproblem," *Computer J.*, vol. 18, no. 1, pp. 74–76, 1975.

[30] J. C. Nash, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Bristol, U.K.: A. Hilger, 1979.
[31] J. K. Salisbury and J. J. Craig, "Articulated hands: Force control and kinematic issues," *Int. J. Robotics Res.*, vol. 1, no. 1, pp. 4–12, 1982.
[32] D. E. Schimmel and F. T. Luk, "A new systolic array for the singular value decomposition," in *Adv. Res. in VLSI*, C. E. Leiserson, Ed. Cambridge, MA: MIT Press, 1986, pp. 205–217.
[33] L. Sciavicco and B. Siciliano, "A solution algorithm to the inverse kinematic problem for redundant manipulators," *IEEE J. Robotics Autom.*, vol. 4, no. 4, pp. 403–410, 1988.
[34] H. Seraji, "Configuration control of redundant manipulators: Theory and implementation," *IEEE Trans. Robotics Autom.*, vol. 5, no. 4, pp. 472–490, 1989.
[35] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," *J. Intell. Robotic Syst.*, vol. 3, pp. 201–213, 1990.
[36] I. D. Walker and J. R. Cavallaro, "Parallel VLSI architectures for real-time control of redundant robots," in *Robotics and Remote Systems: Proc. Fourth Amer. Nuclear Soc. Topical Mtg. Robotics and Remote Syst.*, Feb. 24–28, 1991, pp. 299–309.
[37] C. W. Wampler II, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Trans. Syst., Man, and Cybernetics*, vol. 16, pp. 93–101, 1986.
[38] P. A. Wedin, "Perturbation bounds in connection with singular value decomposition," *BIT*, vol. 12, pp. 99–111, 1972.
[39] T. Yoshikawa, "Analysis and control of robot manipulators with redundancy," in *Robotics Res.: First Int. Symp.*, M. Brady and R. Paul, Ed. Cambridge, MA: MIT Press, 1984, pp. 735–747.
[40] T. Yoshikawa, "Dynamic manipulability of robot manipulators," *J. Robotic Syst.*, vol. 2, no. 1, pp. 113–124, 1985.
[41] T. Yoshikawa, "Manipulability of robotic mechanisms," *Int. J. Robotics Res.*, vol. 4, no. 2, pp. 3–9, Summer 1985.



Anthony A. Maciejewski (S'82–M'87) received the B.S.E.E., M.S., and Ph.D. degrees in electrical engineering from The Ohio State University, Columbus, in 1982, 1984, and 1987, respectively.

Since 1988 he has been with the School of Electrical Engineering at Purdue University, West Lafayette, IN, where he is currently an Associate Professor. His primary research interests center on the simulation and control of kinematically redundant robotic systems.



J. Michael Reagin (S'87–M'91) received the B.S. degree in computer and electrical engineering in 1989 and the M.S. degree in electrical engineering in 1991 from Purdue University, West Lafayette, IN.

He is currently employed by Inland Fisher Guide Division, General Motors, Anderson, IN as a toolroom supervisor responsible for CNC programming and machine operations. His current areas of interest include automatic cutter path generation, graphical simulation, and advanced manufacturing technologies.

Mr. Reagin is a member of Eta Kappa Nu, Tau Beta Pi, and the Society of Manufacturing Engineers.