View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Purdue E-Pubs

Purdue University
Purdue e-Pubs

Department of Electrical and Computer Engineering Technical Reports

Department of Electrical and Computer Engineering

2-1-1987

A CORDIC-Based Pipelined Architecture for Direct Kinematic Position Computation

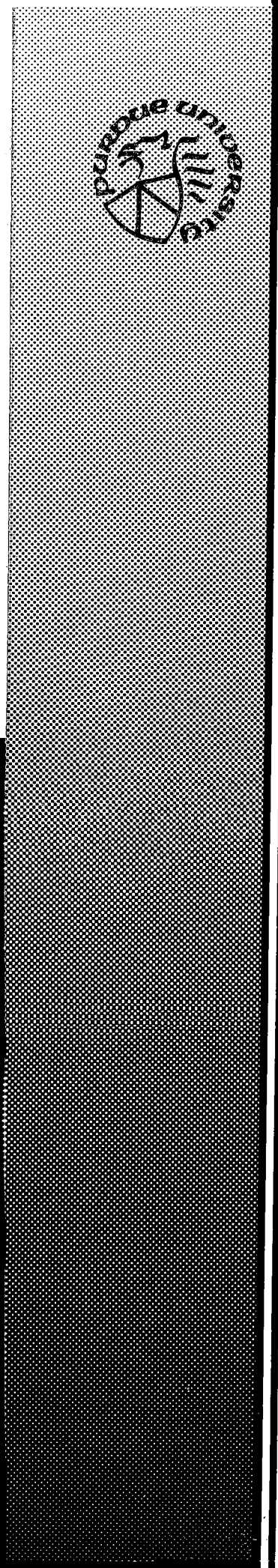
C. S. G. Lee
Purdue University

C. L. Chen
Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/ecetr

Lee, C. S. G. and Chen, C. L., "A CORDIC-Based Pipelined Architecture for Direct Kinematic Position Computation" (1987). *Department of Electrical and Computer Engineering Technical Reports.* Paper 553.
https://docs.lib.purdue.edu/ecetr/553

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.



A CORDIC-Based Pipelined Architecture for Direct Kinematic Position Computation

C. S. G. Lee
C. L. Chen

TR-EE 87-2
February 1987

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

A CORDIC-Based Pipelined Architecture For Direct Kinematic Position Computation

C. S. G. Lee and C. L. Chen

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

TR-EE-87-2

February 1987

ABSTRACT

The kinematic equation of an n-link manipulator involves the chain product of n homogeneous link transformation matrices and reveals the requirement for computing a large set of elementary operations: multiplications, additions, and trigonometric functions. However, these elementary operations, in general, cannot be efficiently computed in general-purpose uniprocessor computers. The CORDIC (COordinate Rotation DIgital Computer) algorithms are the natural candidates for efficiently computing these elementary operations and the interconnection of these CORDIC processors to exploit the great potential of pipelining provides a better solution for computing the direct kinematics. This paper describes a novel CORDIC-based pipelined architecture for the computation of direct kinematic position solution based on the decomposition of the homogeneous link transformation matrix. It is found that a homogeneous link transformation matrix can be decomposed into a product of two matrices, each of which can be computed by two CORDIC processors arranged in parallel, forming a 2-stage cascade CORDIC computational module. Extending this idea to an n-link manipulator, n 2-stage CORDIC computational modules, consisting of $4n$ CORDIC processors, can be concatenated to form a pipelined architecture for computing the position and orientation of the end-effector of the manipulator. Since the initial delay time of the proposed pipelined architecture is $80n\mu s$ and the pipelined time is $40\mu s$, the proposed CORDIC-based architecture requires a total computation time of $(80n + 120)\mu s$ for computing the position and orientation of the end-effector of an n-link manipulator.

This work was supported in part by the National Science Foundation Engineering Research Center Grant CDR-8500022. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the funding agency.

1. Introduction

Robot manipulators are highly nonlinear systems and their motion control is usually specified in terms of the Cartesian path traveled by the end-effector in Cartesian coordinates. If the manipulator is moving in a crowded environment, knowing the location (position and orientation) of its end-effector in real time, as well as its intermediate rigid links, will assist it tremendously in negotiating around the obstacles. The problem of computing the location of the end-effector of a manipulator from the measured data of angular displacements of all the joints is known as the direct kinematics (position) problem [1]. Using the Denavit-Hartenberg matrix representation [1], [2] to describe the translational and rotational relationship between adjacent robot links, the direct kinematics problem reduces to the problem of computing a chain of n 4×4 homogeneous link transformation matrices for an n-link manipulator. This paper presents a novel CORDIC-based pipelined architecture for the computation of these n homogeneous link transformation matrices which yields the location of the end-effector of the manipulator.

The chain multiplication of n homogeneous link transformation matrices yields a set of 12 equations, 9 for orientation matrix and 3 for position information. These equations involve a large set of elementary operations: scalar multiplications, scalar additions, and transcendental functions (sine and cosine). However, these elementary operations, in general, cannot be efficiently computed in general-purpose uniprocessor computers. Moreover, in order to achieve a real-time computation of the end-effector location, time-consuming transcendental functions are implemented as table look-up at the expense of the solution accuracy. Other methods for computing the time-consuming transcendental functions such as the trigonometric function chip [4] and the Taylor series expansion [5] with VLSI implementation have also been proposed. These methods tend to optimize a critical bottleneck computation rather than obtaining an efficient computational scheme for the end-effector location based on the functional/data flow of the kinematic equations.

This paper addresses the computational complexity of the direct kinematic position problem and presents a CORDIC-based pipelined architecture for computing the location of the end-effector of an n-link manipulator in $(80n + 120)\ \mu s$. This efficient CORDIC-based pipelined architecture functions as a peripheral device attached to a conventional host computer (or workstation) which provides the measured data of the manipulator's joint displacements. The CORDIC processors are configured and arranged based on the functional/data flow of the kinematic equations. The architecture design problem was tackled in two separate, but coherent, phases of design. First, the 4×4 homogeneous link transformation matrix ${}^{i-1}\mathbf{A}_i$ for link i of a manipulator was examined and decomposed into a product of two matrices, each of which can be efficiently computed and realized by 2 CORDIC processors arranged in parallel, forming a 2-stage cascade CORDIC computational module. Next, for an n-link

manipulator, n 2-stage CORDIC computational modules, consisting of $4n$ CORDIC processors, are concatenated to form a pipelined architecture for computing the n homogeneous link transformation matrices of the manipulator and the outputs from the pipelined architecture yield the location of the end-effector of the manipulator. Since the execution time of a CORDIC processor is about $40\mu s$ [8], the execution time of a 2-stage CORDIC computational module is $80\mu s$ and the initial delay time of the proposed pipelined architecture for an n-link manipulator is $80n\mu s$. Since the pipeline is balanced [9], [10], the pipelined time of the proposed architecture is $40\mu s$.

2. Coordinate Transformation and Kinematic Equation

To describe the translational and rotational relationship between adjacent robot links, an orthonormal link i coordinate frame, ($\mathbf{x}_i$, $\mathbf{y}_i$, $\mathbf{z}_i$), based on the Denavit-Hartenberg matrix representation is assigned to link i. Once the link coordinate system has been established for each link, a homogeneous transformation matrix, ${}^{i-1}\mathbf{A}_i$, can easily be developed relating the ith coordinate frame to the $(i-1)$th coordinate frame. Using the ${}^{i-1}\mathbf{A}_i$ matrix, one can relate a point $\mathbf{p}_i$ at rest in link i and expressed in homogeneous coordinates with respect to the ith coordinate system to the $(i-1)$th coordinate system established at link $(i-1)$ by

$$\mathbf{p}_{i-1} = {}^{i-1}\mathbf{A}_i \ \mathbf{p}_i \tag{1}$$

where $\mathbf{p}_{i-1} = (x_{i-1}\ ,\ y_{i-1}\ ,\ z_{i-1}\ ,\ 1)^T$, $\mathbf{p}_i = (x_i\ ,\ y_i\ ,\ z_i\ ,\ 1)^T$,

$$ {}^{i-1}\mathbf{A}_i = \begin{cases} \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} & ; \text{ for a rotary joint } i \\ \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & 0 \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} & ; \text{ for a prismatic joint } i \end{cases} \tag{2}$$

and the superscript "T" denotes matrix/vector transpose. The homogeneous transformation matrix ${}^0\mathbf{T}_i$, which specifies the position and orientation of the ith coordinate frame with respect to the base coordinate system, is the chain product of successive homogeneous link transformation matrices of ${}^{i-1}\mathbf{A}_i$ expressed as:

$$
{}^0\mathbf{T}_i = {}^0\mathbf{A}_1\ {}^1\mathbf{A}_2 \cdots {}^{i-1}\mathbf{A}_i = \prod_{j=1}^{i} {}^{j-1}\mathbf{A}_j \tag{3}
$$

$$
= \begin{bmatrix} \mathbf{x}_i & \mathbf{y}_i & \mathbf{z}_i & \mathbf{p}_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ; \text{for } i = 1,2, \cdots, n
$$

Specifically, for $i = n$, we obtain the $\mathbf{T}$ matrix, $\mathbf{T} = {}^0\mathbf{T}_n$, which specifies the position and orientation of the end-effector of a manipulator with respect to the base coordinate system. Consider the $\mathbf{T}$ matrix to be of the form:

$$
\mathbf{T} = \begin{bmatrix} \mathbf{x}_n & \mathbf{y}_n & \mathbf{z}_n & \mathbf{p}_n \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{s} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}
$$

and using the six ${}^{i-1}\mathbf{A}_i$ matrices of the PUMA robot arm in [1], the elements of the $\mathbf{T}$ matrix are found to be

$$
\begin{aligned}
n_x &= C_1[C_{23}(C_4C_5C_6 - S_4S_6) - S_{23}S_5C_6] - S_1[S_4C_5C_6 + C_4S_6] \\
n_y &= S_1[C_{23}(C_4C_5C_6 - S_4S_6) - S_{23}S_5C_6] + C_1[S_4C_5C_6 + C_4S_6] \\
n_z &= -S_{23}[C_4C_5C_6 - S_4S_6] - C_{23}S_5C_6
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
s_x &= C_1[-C_{23}(C_4C_5S_6 + S_4C_6) + S_{23}S_5S_6] - S_1[-S_4C_5S_6 + C_4C_6] \\
s_y &= S_1[-C_{23}(C_4C_5S_6 + S_4C_6) + S_{23}S_5S_6] + C_1[-S_4C_5S_6 + C_4C_6] \\
s_z &= S_{23}(C_4C_5S_6 + S_4C_6) + C_{23}S_5S_6
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
a_x &= C_1(C_{23}C_4S_5 + S_{23}C_5) - S_1S_4S_5 \\
a_y &= S_1(C_{23}C_4S_5 + S_{23}C_5) + C_1S_4S_5 \\
a_z &= -S_{23}C_4S_5 + C_{23}C_5
\end{aligned} \tag{7}
$$

$$
\begin{aligned}
p_x &= C_1[d_6(C_{23}C_4S_5 + S_{23}C_5) + S_{23}d_4 + a_3C_{23} + a_2C_2] - S_1(d_6S_4S_5 + d_2) \\
p_y &= S_1[d_6(C_{23}C_4S_5 + S_{23}C_5) + S_{23}d_4 + a_3C_{23} + a_2C_2] + C_1(d_6S_4S_5 + d_2)
\end{aligned} \tag{8}
$$

$$p_z = d_6(C_{23}C_5 - S_{23}C_4S_5) + C_{23}d_4 - a_3S_{23} - a_2S_2$$

where d_i and a_i are known PUMA's link parameters, and $C_i \equiv \cos\theta_i$, $S_i \equiv \sin\theta_i$, $C_{ij} \equiv \cos(\theta_i + \theta_j)$, and $S_{ij} \equiv \sin(\theta_i + \theta_j)$.

An examination of the above kinematic equations (Eqs. (5)-(8)) shows a large set of elementary operations: multiplications, additions, and transcendental functions. One can use an interconnection of microprocessors with co-processors and an appropriate table look-up technique for the transcendental functions to compute the end-effector location. Although this microprocessor-based computing system is widely used in present day robot controllers, it suffers from the solution accuracy, and lacks flexibility and modularity. The solution inaccuracy is due to the table look-up, while the flexibility is due to the need for changing the link coordinate frames of the manipulator if desired. Furthermore, if one wants to obtain the location of the end-effector with respect to a world coordinate frame instead of the robot's base coordinate frame, then an additional homogeneous transformation matrix relating the base coordinate frame to the external world coordinate frame must be included in Eq. (3). Thus, computing a fixed set of equations as in Eqs. (5)-(8) does not present an attractive solution to the real-time computational problem of the end-effector location. A better solution to the problem is to design a computational architecture that will improve solution accuracy, and achieve flexibility and modularity for computing the location of the end-effector of a manipulator in real time. We propose an interconnection of CORDIC processors to form a pipelined computing machine that efficiently computes the direct kinematics solution. This CORDIC-based pipelined architecture is quite flexible and is based on the concatenation of CORDIC computational modules to form a pipelined machine. A CORDIC computational module consists of 4 CORDIC processors for the computation of a homogeneous link transformation matrix ${}^{i-1}\mathbf{A}_i$. An introduction to CORDIC algorithms and processor is given in the next section and the proposed CORDIC-based pipelined architecture is discussed in section 4.

3. CORDIC Algorithms and Processors

The kinematic equations of an n-link manipulator require the computation of a large set of elementary operations: multiplications, additions, and trigonometric functions. However, these elementary operations, in general, cannot be efficiently computed in general-purpose uniprocessor computers. The CORDIC algorithms [3], [6]-[8], are the natural candidates for efficiently computing these elementary operations. They represent an efficient way to compute a variety of functions related to coordinate transformations with iterative procedures involving only shift-and-add operations at each step. Thus, cordic processing elements are extremely simple and quite compact to realize [7], [8] and the interconnection of CORDIC processors to exploit the great potential of pipelining provides a novel solution for computing the direct

kinematic position solution.

To establish connections between cordic and rotation-based algorithms, let the angle of rotation θ be decomposed into a sum of n sub-angles $\{d_i;\ i = 0,\ n-1\}$

$$\theta = \sum_{i=0}^{n-1} u_i d_i \tag{9}$$

where the sign u_i (± 1) is chosen based on the direction of rotation. Similarly, the plane rotation matrix $\mathbf{R}(\theta)$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{10.a}$$

or the hyperbolic rotation matrix $\mathbf{R}(\theta)$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cosh\theta & \sinh\theta \\ -\sinh\theta & \cosh\theta \end{bmatrix} \tag{10.b}$$

can also be decomposed into a product of sub-angle rotation matrices

$$\mathbf{R}(\theta) = \prod_{i=0}^{n-1} \mathbf{R}(d_i) \tag{11}$$

Thus, a single rotation of θ angle can be replaced by n smaller rotations with d_i angle each. In the cordic algorithms, d_i is chosen such that

$$d_i = \begin{cases} \tan^{-1}(2^{-s(i)}) & ,\ m = 1 \text{ (circular)} \\ 2^{-s(i)} & ,\ m = 0 \text{ (linear)} \\ \tanh^{-1}(2^{-s(i)}) & ,\ m = -1 \text{ (hyperbolic)} \end{cases} \tag{12}$$

where $m = (\ = -1, 0, 1)$ determines the type of rotations and $\{s(i);\ i = 0,\ n-1\}$ is a non-decreasing integer sequence. Using d_i from Eq. (12), $\mathbf{R}(d_i)$ can be written as

$$\mathbf{R}(d_i) = p_i \begin{bmatrix} 1 & -m u_i 2^{-s(i)} \\ u_i 2^{-s(i)} & 1 \end{bmatrix} \tag{13}$$

where p_i is a scaling factor and equals to $(1 + m\, 2^{-2s(i)})^{-1/2}$. Let $\mathbf{R}^N(\theta)$ and $\mathbf{R}^N(d_i)$ be the normalized form of $\mathbf{R}(\theta)$ and $\mathbf{R}(d_i)$, respectively, then from Eq. (11), we have

$$\mathbf{R}(\theta) = \prod_{i=0}^{n-1} p_i \prod_{i=0}^{n-1} \mathbf{R}^N(d_i) \tag{14.a}$$

$$= k_m \prod_{i=0}^{n-1} \mathbf{R}^N(d_i) = k_m\, \mathbf{R}^N(\theta) \tag{14.b}$$

where

$$k_m = \prod_{i=0}^{n-1} p_i = \prod_{i=0}^{n-1} (1 + m\, 2^{-2s(i)})^{-1/2} \tag{14.c}$$

and

$$\mathbf{R}^N(\theta) = \prod_{i=0}^{n-1} \begin{bmatrix} 1 & -m u_i 2^{-s(i)} \\ u_i\ 2^{-s(i)} & 1 \end{bmatrix} \tag{14.d}$$

Usually, k_m is a machine constant and $k_m \simeq 0.6072$ (for $m = 1$) or 1.00 (for $m = 0$) or 1.205 (for $m = -1$), when $n \geq 10$ [6], [11]. The normalized rotation matrix of Eq. (14.d) indicates that each small rotation can be realized with one simple shift-and-add operation. Hence, the computation of a trigonometric function can be accomplished with n shift-and-add operations, which is comparable to conventional multiplications. This makes a CORDIC ALU a very appealing alternative to the traditional ALU for implementing the elementary functions. In general, the normalized CORDIC algorithm can be written as follows:

FOR $i = 0, 1, \cdots, n-1$, DO

$$\begin{bmatrix} x_{i+1}^N \\ y_{i+1}^N \end{bmatrix} = \begin{bmatrix} 1 & -m\ u_i\ 2^{-s(i)} \\ u_i\ 2^{-s(i)} & 1 \end{bmatrix} \begin{bmatrix} x_i^N \\ y_i^N \end{bmatrix} \tag{15.a}$$

$$z_{i+1}^N = z_i^N + u_i\ d_i \tag{15.b}$$

where $x_0^N = x_0$, $y_0^N = y_0$, m determines the type of rotation, d_i is chosen as in Eq. (12), and the auxiliary variable z_i^N is introduced to accumulate the rotation after each iteration. And the corresponding "unnormalized" CORDIC algorithm is described as:

FOR $i = 0, 1, \cdots, n-1$, DO

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = p_i \begin{bmatrix} 1 & -m\ u_i\ 2^{-s(i)} \\ u_i\ 2^{-s(i)} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{16.a}$$

$$z_{i+1} = z_i + u_i d_i \tag{16.b}$$

where $x_0 = x_0$ and $y_0 = y_0$. It can be shown that z_i and z_i^N will accumulate the angle of the total rotation and have the same value after n iterations. However, the end results of (x_n, y_n) from the iterations of Eq. (16.a) and the end results of (x_n^N, y_n^N) from the iterations of Eq. (15.a) are related according to

$$x_n = k_m\ x_n^N \quad ; \quad y_n = k_m\ y_n^N \tag{17}$$

Consequently, one may evaluate x_n^N and y_n^N by using only the shift-and-add operations

in Eq. (16.a), then realize x_n and y_n by other simple methods such as ROM look-up tables and regular combinatorial logic, etc. Fortunately, it is possible to find a simple way to normalize the scale factor k_m using the same shift-and-add hardware [8], [11]. The supplementary operations that are used to force the scale factor k_m to converge toward unity can be either performed after all the operations of Eq. (15.a) are terminated, that is,

$$x^c_{i+1} = (1 + \gamma_i 2^{-i})\, x^c_i \tag{18}$$

$$y^c_{i+1} = (1 + \gamma_i\, 2^{-i}) y^c_i$$

where $x^c_0 = x^N_n$, $y^c_0 = y^N_n$, and $0 \leq i \leq n-1$, or interleaved with the operations of Eq. (15.a), that is,

$$x^c_i = (1 + \gamma_i\, 2^{-i})\, x^N_i \tag{19}$$

$$y^c_i = (1 + \gamma_i\, 2^{-i})\, y^N_i$$

where $0 \leq i \leq n-1$. The parameter γ_i in Eq. (18) or Eq. (19) may be -1 or 0 or 1 depending on the value of i and the type of rotations (i.e. m) [8], [11].

Haviland et al. [8] realized the CORDIC algorithm on a CMOS chip and showed that the processing time of the CORDIC chip is 40 μs. They also showed the SPICE analysis [12] of the chip and suggested $n = 13$ as the minimum cycle time of a two-byte (24-bit) fixed-point operation. However, in practice, they used $n = 24$. For a conventional CORDIC module, it requires 5 shift-and-add modules to compute one CORDIC iteration and one normalization iteration in parallel (that is, 3 shift-and-add modules for Eqs. (15.a) and (15.b), and 2 shift-and-add modules for Eq. (19)). The desired output can be obtained in 24 iterations ($n = 24$). Thus, 24 iterations of 5 shift-and-add modules computing in parallel will be enough to realize CORDIC algorithms. This indicates that the CORDIC processing time is no slower than the time for a serial multiplier computing two 24-bit operands.

It is possible to enhance the throughput of a conventional CORDIC module by using a pipelined CORDIC module or a doubly-piped CORDIC module [11]. For example, if a conventional CORDIC module requires $T = n\, T_c$ time to complete the computation (where T_c is the time for one CORDIC iteration of Eq. (15) and one normalization iteration of Eq. (19)), where n is the number of iterations, it is suggested that a CORDIC pipe consists of a cascade of n layers and each layer has 3 shift-and-add processing elements computing one CORDIC iteration of Eq. (15) in parallel, then 2 shift-and-add processing elements can compute one normalization iteration of Eq. (19) in parallel. Thus, the CORDIC pipe is operating at an effective rate of $T_c^{'}$ per operation rather than $n\, T_c$, where $T_c^{'}$ is the time for one shift-and-add operation which is always less than T_c. Furthermore, a doubly-piped CORDIC module has been introduced [11] to operate at bit level and has a throughput of one sample per clock

period. This enhancement will truly improve the performance and throughput of the proposed CORDIC-based architecture.

Figure 1 summarizes the elementary functions that can be obtained from the CORDIC processor when m is set to -1, 0, or 1. In this figure, a CORDIC processor is depicted as a box with three inputs x_0 , y_0 , z_0, which are the initial values of x_i , y_i, and z_i in Eq. (16), as well as three outputs that correspond to the final values of x_n , y_n , and z_n in Eq. (16). Thus, the outputs x_n , y_n , z_n are the desired elementary functions, when m is appropriately set to -1, 0, or 1. These CORDIC processors will be configured and connected, based on the decomposition of the homogeneous link transformation matrices of a manipulator, to arrive at an efficient CORDIC-based pipelined architecture for the computation of the direct kinematic position solution.

4. CORDIC-Based Pipelined Architecture

The design philosophy of the proposed CORDIC-based pipelined architecture is to examine the direct kinematic position solution for its computational flow and data dependencies in order to functionally decompose the computations into a cascade of CORDIC computational modules (CCMs) with an objective that each CCM will be realizable by CORDIC processors. For an n-link manipulator, the kinematic equations reveal the chain product of successive homogeneous link transformation matrices of ${}^{i-1}\mathbf{A}_i$, $i = 1, 2, \cdots, n$. Thus, we first look at the possible decomposition of the link i homogeneous transformation matrix ${}^{i-1}\mathbf{A}_i$. For the link i homogeneous transformation matrix ${}^{i-1}\mathbf{A}_i$, it can be decomposed as a product of four basic homogeneous translation/rotation matrices as [1],

$$ {}^{i-1}\mathbf{A}_i = Tran(\mathbf{z}_{i-1}, d_i)\, Rot(\mathbf{z}_{i-1}, \theta_i)\, Tran(\mathbf{x}_i, a_i)\, Rot(\mathbf{x}_i, \alpha_i) \tag{20} $$

$$ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_i & -sin\,\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Or, from Eq. (20), ${}^{i-1}\mathbf{A}_i$ can be decomposed as a product of two matrices as,

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -sin\,\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{21} $$

Since as stated earlier, the homogeneous link transformation matrix ${}^{i-1}\mathbf{A}_i$ is used to

transform a vector expressed in the ith coordinate frame to the same vector expressed in the $(i-1)$th coordinate frame, this coordinate transformation can be performed in two sequential steps as indicated in Eq. (21). That is, the first step is transforming a vector $\mathbf{p}_i = (x_i, y_i, z_i, 1)^T$ in the ith coordinate frame to an intermediate vector $\mathbf{x}_i^A = (x_i^A, y_i^A, z_i^A, 1)^T$,

$$\begin{bmatrix} x_i^A \\ y_i^A \\ z_i^A \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} x_i + a_i \\ y_i C\alpha_i - z_i S\alpha_i \\ z_i C\alpha_i + y_i S\alpha_i \\ 1 \end{bmatrix} \tag{22}$$

and the second step is to map the intermediate vector $\mathbf{x}_i^A$ to the desired vector $\mathbf{p}_{i-1} = (x_{i-1}, y_{i-1}, z_{i-1}, 1)^T$,

$$\begin{bmatrix} x_{i-1} \\ y_{i-1} \\ z_{i-1} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -sin\,\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i^A \\ y_i^A \\ z_i^A \\ 1 \end{bmatrix} = \begin{bmatrix} x_i^A C\theta_i - y_i^A S\theta_i \\ y_i^A C\theta_i + x_i^A S\theta_i \\ z_i^A + d_i \\ 1 \end{bmatrix} \tag{23}$$

Looking at the elementary functions computed by CORDIC processors in Figure 1, Eq. (22) can be computed and realized by two CORDIC processors arranged in parallel as follows:

Step 1—a: CORDIC Processor: CIRC 1

$$\text{Input} = \begin{cases} x_0 = y_i \\ y_0 = z_i \\ z_0 = \alpha_i \end{cases}$$

$$\text{Output} = \begin{cases} x1_n = y_i C\alpha_i - z_i S\alpha_i \equiv y_i^A \\ y1_n = z_i C\alpha_i + y_i S\alpha_i \equiv z_i^A \\ z1_n = \text{Not used} \end{cases}$$

Step 1—b: CORDIC Processor: LIN 1

$$\text{Input} = \begin{cases} x_0 = 1 \\ y_0 = a_i \\ z_0 = x_i \end{cases}$$

$$\text{Output} = \begin{cases} x2_n = \text{Not used} \\ y2_n = x_i + a_i \equiv x_i^A \\ z2_n = \text{Not used} \end{cases}$$

Note that steps 1-a and 1-b are computed in parallel. Similarly, Eq. (23) can be computed and realized by two CORDIC processors arranged in parallel as follows:

Step 2—a: CORDIC Processor: CIRC 1

$$\text{Input} = \begin{cases} x_0 = y2_n \equiv x_i^A \\ y_0 = x1_n \equiv y_i^A \\ z_0 = \theta_i \end{cases}$$

$$\text{Output} = \begin{cases} x3_n = x_i^A C\theta_i - y_i^A S\theta_i \equiv x_{i-1} \\ y3_n = y_i^A C\theta_i + x_i^A S\theta_i \equiv y_{i-1} \\ z3_n = \text{Not used} \end{cases}$$

Substituting x_i^A and y_i^A from Eq. (22) into the above output equations, we have

$$\text{Output} = \begin{cases} x3_n = x_i C\theta_i - y_i C\alpha_i S\theta_i + z_i S\alpha_i S\theta_i + a_i C\theta_i \equiv x_{i-1} \\ y3_n = x_i S\theta_i + y_i C\alpha_i C\theta_i - z_i S\alpha_i C\theta_i + a_i S\theta_i \equiv y_{i-1} \\ z3_n = \text{Not used} \end{cases}$$

Step 2—b: CORDIC Processor: LIN 1

$$\text{Input} = \begin{cases} x_0 = 1 \\ y_0 = d_i \\ z_0 = y1_n \equiv z_i^A \end{cases}$$

$$\text{Output} = \begin{cases} x4_n = \text{Not used} \\ y4_n = z_i^A + d_i \equiv z_{i-1} \\ z4_n = \text{Not used} \end{cases}$$

Substituting z_i^A from Eq. (22) into the above output equations, we have

$$\text{Output} = \begin{cases} x4_n = \text{Not used} \\ y4_n = y_i S\alpha_i + z_i C\alpha_i + d_i \equiv z_{i-1} \\ z4_n = \text{Not used} \end{cases}$$

Note that the outputs in steps 2-a and 2-b (i.e. $x3_n$, $y3_n$, and $y4_n$) correspond to the result of the matrix-vector multiplication in Eq. (1).

Since the outputs of the CORDIC processors in steps 1-a and 1-b are fed into the inputs of the CORDIC processors in steps 2-a and 2-b, the interconnection of these four CODIC processors forms a 2-stage cascade CORDIC computational module (CCM) for computing a general homogeneous link transformation matrix ${}^{i-1}\mathbf{A}_i$ of a manipulator. This 2-stage cascade CORDIC computational module is shown in Figure 2. Extending this idea to an n-link manipulator, we need to cascade n CCMs, consisting of $4n$ CORDIC processors, to form a pipelined architecture for computing the n homogeneous link transformation matrices in the kinematic equation, and the outputs of this pipelined architecture transform the vector $\mathbf{p}_n$ expressed in the nth coordinate frame to the same vector expressed in the base coordinate frame of the manipulator. Since the vector $\mathbf{p}_n$ is chosen arbitrarily, if we let $\mathbf{p}_n = (0,0,0,1)^T$, then the output of the proposed CORDIC-based pipelined architecture is the position of the origin of the link n coordinate frame with respect to the base coordinate frame. Similarly, letting $\mathbf{p}_n = (1,0,0,1)^T$, $\mathbf{p}_n = (0,1,0,1)^T$, and $\mathbf{p}_n = (0,0,1,1)^T$, we, respectively, obtain the orientation (normal, sliding, and approach vectors) of the link n coordinate frame with respect to the base coordinate frame. Thus, in order to obtain the location of the end-effector of a manipulator, we need to pipe a set of 4 input vectors (or a 4×4 identity matrix) into the proposed pipelined architecture to obtain the $[\mathbf{n},\mathbf{s},\mathbf{a},\mathbf{p}]$. For a PUMA robot arm in [1], where $n = 6$, a pipelined architecture of 6 CORDIC computational modules with 24 CORDIC processors can be used to compute the kinematic equation in Eq. (3) and is shown in Figure 3.

Several key features and characteristics about this CORDIC-based pipelined architecture should be addressed and discussed:

(1) *Flexibility.* The 2-stage CORDIC computational module shown in Figure 2 computes a general homogeneous link transformation matrix ${}^{i-1}\mathbf{A}_i$. Thus, the CCM is suitable for any manipulator (with prismatic or rotary joints) whose link coordinate frames are described by 4×4 homogeneous transformation matrices. The inputs to the CORDIC processors in the CCM are link/joint parameters (i.e. d_i , θ_i , a_i , α_i) of link/joint i and a vector $\mathbf{p}_i = (x_i , y_i , z_i , 1)^T$ expressed in the ith coordinate frame. Thus, changing the link coordinate frames of the manipulator will only affect the input values of the CORDIC processors and will not alter the structure of the CCM and the proposed pipelined architecture.

(2) *Modularity.* The idea of using 4 CORDIC processors to form a 2-stage cascade CORDIC computational module provides a modular approach in designing our proposed pipelined architecture. This modularity is based on the characteristics of the kinematic equation which involves the computation of the chain product of n homogeneous link transformation matrices for an n-link manipulator. Each CCM becomes a building block (or computational block) for computing one of the

n

homogeneous link transformation matrices in the kinematic equation in Eq. (3). If one wants to relate the link n coordinate frame to an external world coordinate frame instead of the robot's base coordinate frame, then an additional CORDIC computational module can be appropriately put in cascade with the existing n CCMs. This additional CCM computes the homogeneous transformation matrix which relates the base coordinate frame to the external world coordinate frame. Similarly, an additional CCM can be appropriately cascaded into the existing pipeline to compute the homogeneous transformation matrix which relates the tool coordinate frame to the link n coordinate frame of the manipulator. This homogeneous-transformation-matrix-based CORDIC module concept changes the building block of our architecture from CORDIC processors to CCMs. Thus, the number of CCMs in the pipelined architecture directly corresponds to the number of homogeneous link transformation matrices in the kinematic equation of the manipulator.

(3) *Solution Accuracy.* As indicated in [8], CORDIC algorithms were realized on a CMOS chip with 24-bit data processing. Based on fixed-point arithmetic, the iterative algorithm converges with an error of 2^{-24}. This solution accuracy is much better than those quoted in [4], [5].

(4) *Computational Time.* In developing the CORDIC computational module, we used an arbitrary vector $\mathbf{p}_i$ expressed in link i coordinate frame as an input to the ith CORDIC computational module. If we extend this idea to the link n coordinate frame for an n-link manipulator and let $\mathbf{p}_n = (0,0,0,1)^T$, then the output of this pipeline is the position of the origin of the link n coordinate frame with respect to the base coordinate frame. Similarly, letting $\mathbf{p}_n = (1,0,0,1)^T$, $\mathbf{p}_n = (0,1,0,1)^T$, and $\mathbf{p}_n = (0,0,1,1)^T$, we, respectively, obtain the orientation (i.e. $[\mathbf{n},\mathbf{s},\mathbf{a}]$) of the link n coordinate frame with respect to the base coordinate frame. Since a reasonable execution time for a CORDIC processor is 40 μs, the processing time of a 2-stage cascade CCM is $80\mu s$. For a pipelined architecture consisting of n CORDIC computational modules (with $4n$ CORDIC processors) for an n-link manipulator, the initial delay time in the pipeline is $80n\,\mu s$ and the pipelined time is $40\mu s$. Thus, in order to obtain the position and orientation of the end-effector of a manipulator, we need to pipe a set of 4 input vectors (or a 4x4 identity matrix) into the proposed pipelined architecture to obtain the $[\mathbf{n},\mathbf{s},\mathbf{a},\mathbf{p}]$. The first output from this set of 4 input vectors will take an initial delay time of $80n\,\mu s$, then the successive outputs will be $40\mu s$ apart because of the pipelined architecture. This gives a total computation time of $(80n + 120)\mu s$ for computing the position and orientation of the end-effector for an n-link manipulator. For a PUMA robot arm, the computation time for obtaining the location of the end-effector is the initial delay time (for position) plus 3 pipelined time (for $[\mathbf{n},\mathbf{s},\mathbf{a}]$) for a total of $600\mu s$. The proposed

pipelined architecture consists of 6 CCMs with 24 CORDIC processors.

5. Conclusion

The kinematic equation of an n-link manipulator requires the computation of n homogeneous link transformation matrices. The decomposition of a homogeneous link transformation matrix into a product of two matrices reveals that the computation of the homogeneous transformation matrix can be accomplished by a 2-stage CORDIC computational module consisting of 4 CORDIC processors. Thus, n 2-stage CORDIC computational modules, consisting of $4n$ CORDIC processors, can be cascaded together to form a CORDIC-based pipelined architecture for computing the position and orientation of the end-effector of the manipulator. The proposed pipelined architecture with a cascade of n CORDIC computational modules has an initial delay time of $80n\,\mu s$ and a pipelined time of $40\mu s$. The CORDIC-based pipelined architecture requires a total computation time of $(80n + 120)\mu s$ for computing the position and orientation of the end-effector of an n-link manipulator. For a PUMA robot arm, given the measured data of the angular displacements of all the joints and the link/joint parameters of the robot, the computation time for locating the end-effector is $600\mu s$ and the pipelined architecture consists of 6 CCMs with 24 CORDIC processors.

6. References

1. K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence,* McGraw-Hill, Chapter 2, September 1986.

2. J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics,* pp. 215-221, 1955.

3. J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electronic Computers,* vol. EC-8, no. 3, pp. 330-334, Sept. 1959.

4. S. S. Leung and M. A. Shanblatt, "A VLSI Chip Architecture for the Real-Time Computation of Direct Kinematics," *Proc. of 1986 IEEE Int'l Conf. on Robotics and Automation,* San Francisco, CA, pp. 1717-1722, April 1986.

5. V. Seshadri, "A Real-Time VLSI Architecture for Direct Kinematics," *Proc. of 1987 IEEE Int'l Conf. on Robotics and Automation,* Raleigh, NC, March 30 - April 3, 1987.

6. J. S. Walther, "A Unified Algorithm for Elementary Function," *AFIPS Conf. Proc.,* vol. 38, 1971 SJCC, pp. 379-385.

7. H. M. Ahmed, J. M. Delosme and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *IEEE Computer,* vol. 15, no. 1, pp. 65-82, Jan. 1982.

8. G. L. Haviland and A. A. Tuszynski, "A CORDIC Arithmetic Processor Chip," *IEEE Trans. Comput.,* vol. C-29, no. 2, pp. 68-79, Feb. 1980.

9. P. M. Kogge, *The Architecture of Pipelined Computers,* McGraw-Hill, New York, 1981.

10. C. S. G. Lee and P. R. Chang, "A Maximum Pipelined CORDIC Architecture for Robot Inverse Kinematics Computation," School of Electrical Engineering, Technical Report TR-EE 86-5, Purdue University, January 1986.

11. P. Dewide et al., "Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms," in *VLSI and Modern Signal Processing,* S. Y. Kung, H. J. Whitehouse, T. Kailath, (eds.), Prentice-Hall, Inc., Englewood Cliffs, NJ, pp. 257-276.

12. L. W. Nagel, "Spice2: A computer program to simulate semiconductor circuits," Univ. of California, Berkeley, ELR, 1975.

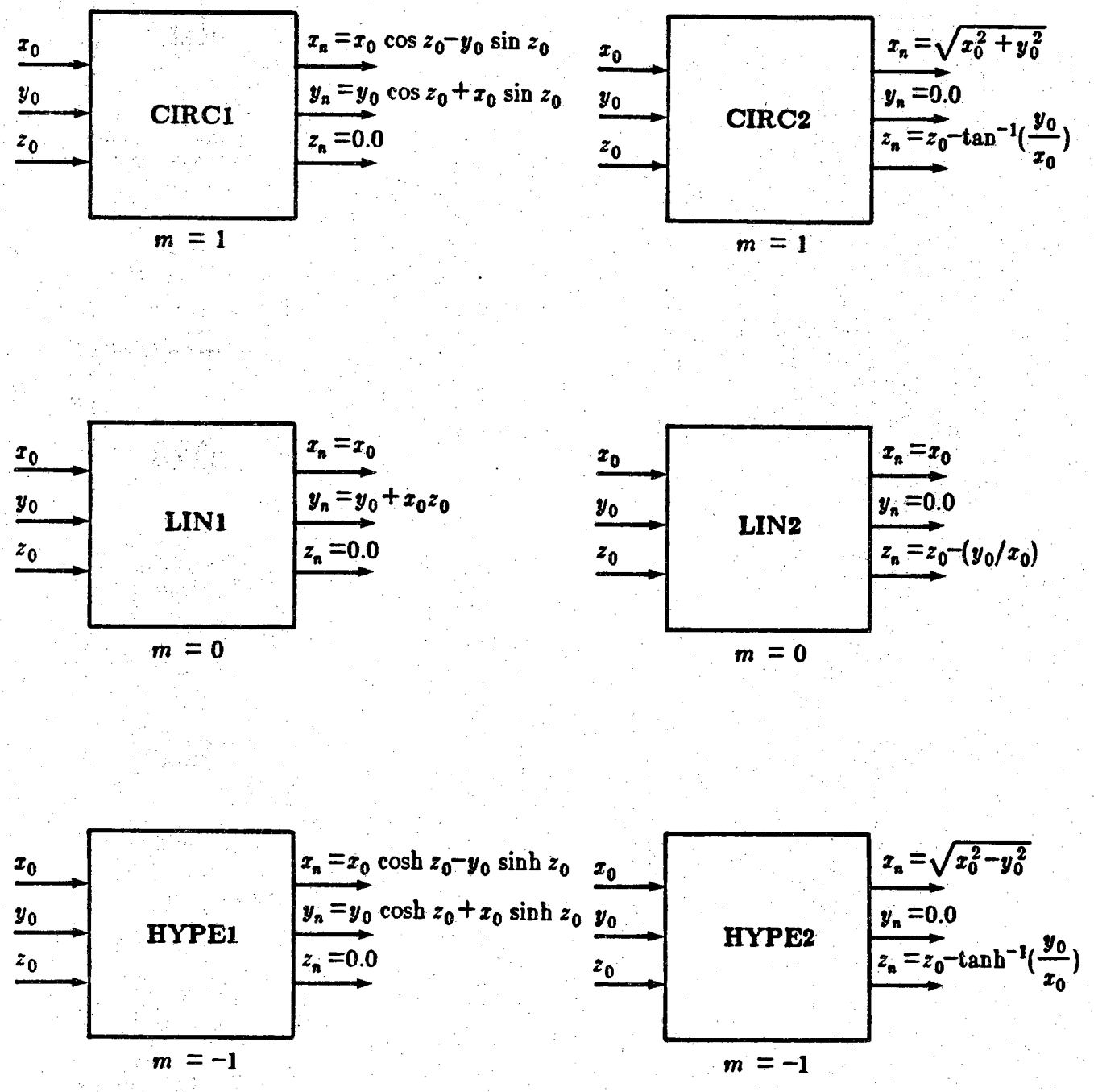


Figure 1 Elementary Functions Computed by CORDIC Processors

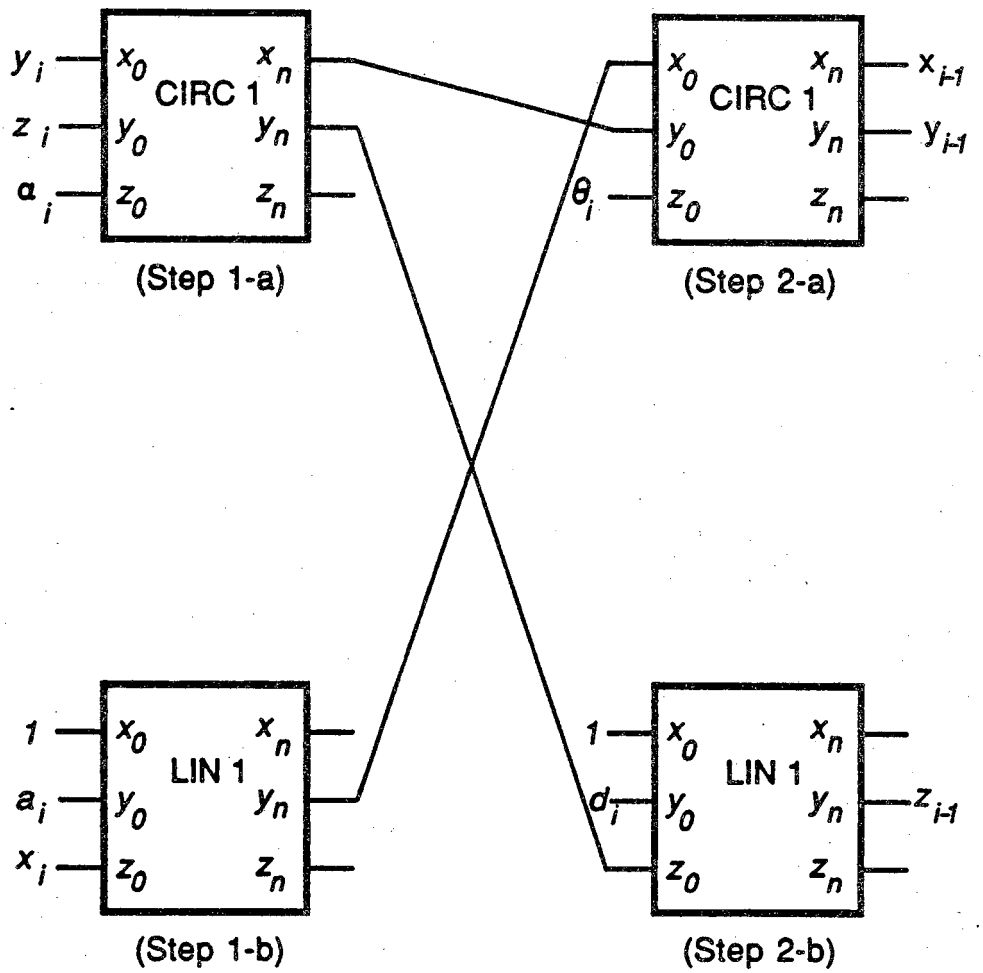


Figure 2 A 2-Stage CORDIC Computational Module for Computing ${}^{i-1}\mathbf{A}_i$

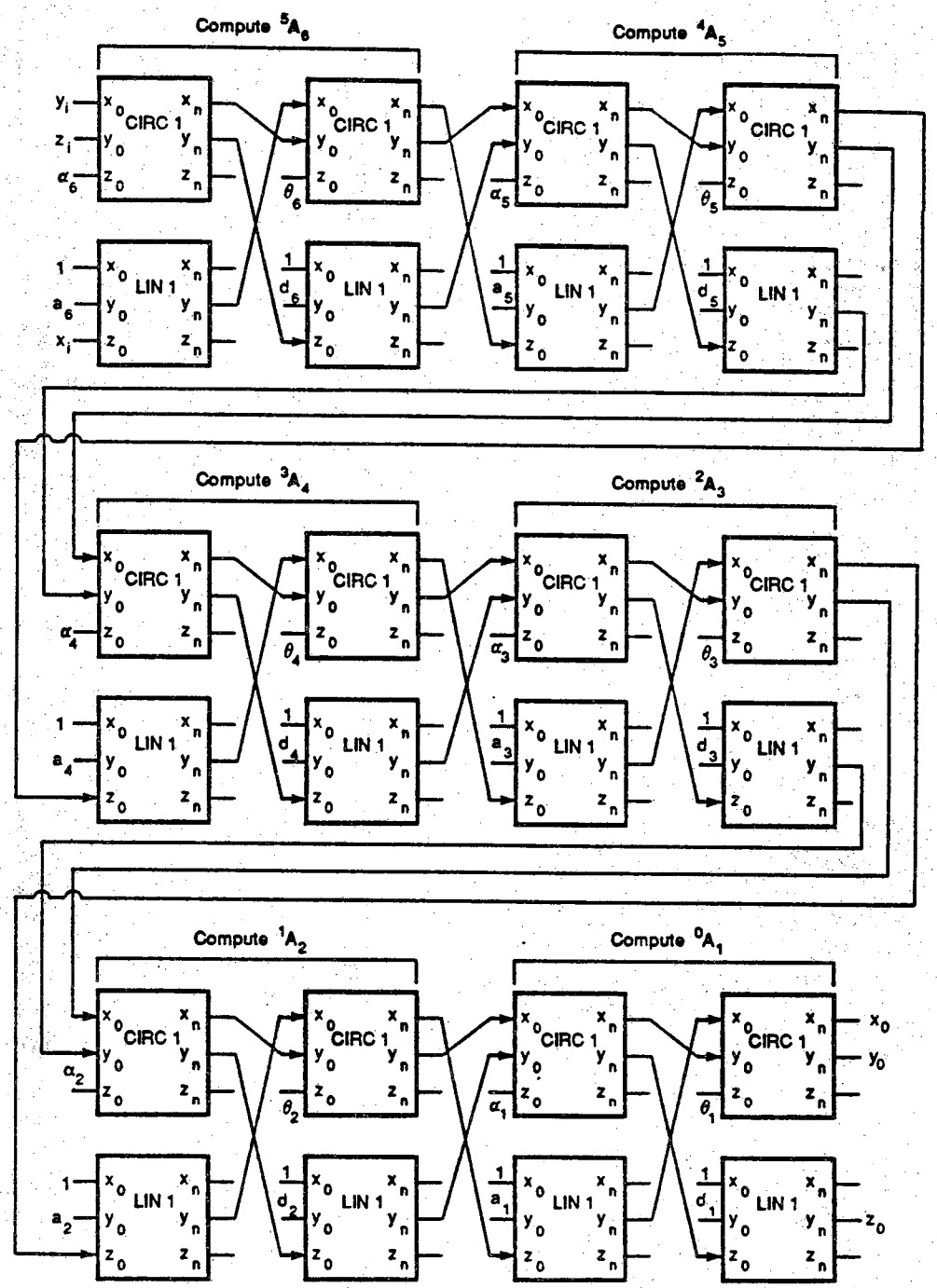


Figure 3 A CORDIC-Based Pipelined Architecture for Direct Kinematics Computation. $\mathbf{p}_n = (x_i, y_i, z_i)^T$ is the input vector and $\mathbf{p}_0 = (x_0, y_0, z_0)^T$ is the output vector.
When $\mathbf{p}_n = (0,0,0)^T$, $\mathbf{p}_0$ is the position vector of the end-effector.
When $\mathbf{p}_n = (1,0,0)^T$, $\mathbf{p}_0$ is the normal vector of the end-effector.
When $\mathbf{p}_n = (0,1,0)^T$, $\mathbf{p}_0$ is the sliding vector of the end-effector.
When $\mathbf{p}_n = (0,0,1)^T$, $\mathbf{p}_0$ is the approach vector of the end-effector.