基於可變因子CORDIC新演算法的座標轉換
NEW CORDIC-BASED COORDINATE TRANSFORMATION ALGORITHMS USING VARIABLE FACTORS*

林秀峰　　羅浩榮　　莊作彬
Hsiu-Feng Lin, Hao-Yung Lo, Tso-Bing Chang

逢甲大學資訊工程研究所
Institute of Information Engineering, Feng Chia University
Wen-Hwa Road 100, Taichung

摘要

本篇論文中，描述了採用變動因子的CORDIC新演算法的座標轉換。由對此演算法所作的模擬中發現，僅須最多12次的迴圈便可獲得高達23位元的精度，因爲每次迴圈使用了帶號數2位元以取代1位元之故。模擬的結果中亦可得知精確度的範圍是界於96%-99%之間，其它誤差最多只有1位元即2^{-23}誤差。除此之外，本演算無需進行耗時的乘、除法卻依然保有傳統CORDIC演算法的基本概念。

關鍵詞: CORDIC 演算法，計算機算術

Abstract

In this paper we will describe our new CORDIC algorithms which are applied in coordinate transformation using variable factors. In our simulation for our new algorithms. the number of iterations is 12 maximum. with the accuracy of 23 bit length. This is accomplished by using signed two bits. instead of one to process at a time. Simulation results show that the accuracy can be up to 99% maximum and 96% minimum. Besides. our new algorithms are multiplication-free and division-free and keep the original concept of the classical CORDIC algorithms.

Keywords: *CORDIC, computer arithmetic, algorithm.*

1. Introduction

Our research proposed another CORDIC algorithms to perform the coordinate transformation. Yang[6]. used variable factors to perform 16-Bit sine and cosine computation with over 91% accuracy. This paper is organized as follows. The second section of this paper gives a brief survey of CORDIC . Our new algorithm that makes use of ***variable factors*** for performing plane rotation is described in the third section. It achieved over 95% accuracy as compared to the results obtained by C language standard library. In the fourth section we will describe our new algorithm which performs the transformation between plane coordinate and polar coordinate. This algorithm we propose can get lower iteration time and higher accuracy. thus producing an efficient execution of CORDIC algorithm. The last section is our conclusion for this paper.

2. A Brief Survey of CORDIC

Consider the computation of plane rotation of an given angle θ (rad). based on CORDIC algorithm[8]. tthe following iterations should be executed.

$$x_{i-1} = x_i - d_i \cdot y_i \cdot 2^{-i} \quad \text{.....(2-1)}$$

$$y_{i-1} = y_i + d_i \cdot x_i \cdot 2^{-i} \quad \text{.....(2-2)}$$

$$z_{i-1} = z_i - d_i \cdot \alpha_i \quad \text{.....(2-3)}$$

where $d_i \in \{-1,1\}, \alpha_i = \arctan 2^{-i}$.

If we order i=0.1.2.....(n-1). we can get the result of the generalized equations.

$$x_n = K\left[x_0 \cos\alpha - y_0 \sin\alpha\right] \quad \text{.....(2-4)}$$

$$y_n = K\left[y_0 \cos\alpha + x_0 \sin\alpha\right] \quad \text{.....(2-5)}$$

$$z_n = z_0 - \alpha \quad \text{.....(2-6)}$$

where

$$\alpha = \sum_{i=0}^{n-1} d_i \alpha_i \quad \text{.....(2-7)}$$

*(本篇曾或獲 1995 年全錄論文獎)

$$K = \prod_{i=0}^{n-1}\left(1+2^{-2i}\right)^{1/2} = \prod_{i=0}^{n-1} K_i \quad \text{.....(2-8)}$$

Where K is called the scaling factor which is converge to a constant 1.6468. In the classic CORDIC algorithm, d_i is selected in each step so that the net sum of angle z_n approaches to zero. If we initially set $x_0 = x/K, y_0 = y/K$, and $z_0 = \theta$, then $x' = x\cos\theta - y\sin\theta$ and $y' = x\sin\theta + y\cos\theta$ are obtained simultaneously from x_n and y_n, respectively. Note that the operation of the above equations can be easily executed by shift, addition/subtraction. The values of $\alpha_i = \arctan 2^{-i}$ are provided by a small look-up table. In the above derivations, we can get the basic CORDIC execution concept. The classic CORDIC algorithm use a constant factor to scale up the final results, producing an execution error and wasting execution time. In the following sections, we will introduce our new CORDIC algorithms which make use of variable factors to replace the classical CORDIC algorithms and show their high accuracy and less iterations.

3. Algorithm of a CORDIC-Based Plane Rotation

Let P(x,y) be a plane vector. Consider the plane rotation of angle θ (rad) of the points P(x,y). Assume that P is transformed into the new vector P'(x',y'), then

$x' = x\cos\theta - y\sin\theta$ ..... (3-1)

$y' = x\sin\theta + y\cos\theta$ ..... (3-2)

Let us see the CORDIC algorithm that performs the above computations.

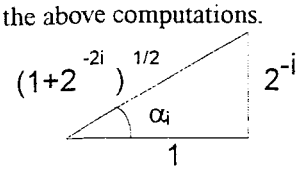


Figure 3.1.A The triangle of α_i

$$\begin{cases} Z_{i+1} = Z_i - q_i\alpha_i \text{.....(3-3)} \\ X_{i+1} = \cos\alpha_i(X_i - q_i Y_i \tan\alpha_i) \text{.....(3-4)} \\ Y_{i+1} = \cos\alpha_i(Y_i + q_i X_i \tan\alpha_i) \text{.....(3-5)} \end{cases}$$

where $q_i \in \{-1,0,1\}$

In order to simplify multiplication, we fix angle $\alpha_i = \tan^{-1} 2^{-i}$, i=0,1,2,.....n-1. After n iterations, we get:

$$X_n = X_0\cos(q_0a_0 + q_1a_1 + ... + q_{n-1}a_{n-1}) - Y_0\sin(q_0a_0 + q_1a_1 + ... + q_{n-1}a_{n-1}) \quad \text{.....(3-6)}$$

$$Y_n = Y_0\cos(q_0\alpha_0 + q_1\alpha_1 + ... + q_{n-1}\alpha_{n-1}) + X_0\sin(q_0\alpha_0 + q_1\alpha_1 + ... + q_{n-1}\alpha_{n-1}) \quad \text{.....(3-7)}$$

If the input angle is θ, $x_0 = x, y_0 = y$, when $Z_n \to 0$, we obtain

$Xn = x\cos\theta - y\sin\theta$ .....(3-8)

$Yn = x\sin\theta + y\cos\theta$ .....(3-9)

3.1 Derivation Process

In the above equations, if we start with x and y, rotation angle θ, then we can get x' and y'. This corresponds in computer graphics that points in the graph rotates, generating a new graph. Since we want to develop a CORDIC-based algorithm that performs this transformation, we take the following steps to derive this new algorithm. The input angle is represented in binary format.

(A)Case 1(The first four bits after the point of the input angle are not all zeros)

3.1.A.1 The definition of angle β:

We now give a definition of the angle β: $\beta_i = 2\alpha_i$, for i=0,1,2,.....n-1, where $\alpha_i = \tan^{-1} 2^{-i}$.

By our definition,

$$\beta_i = 2\alpha_i = 2\tan^{-1} 2^{-i}$$

$$= 2\times[2^{-i} - \frac{1}{3}2^{-3i} + \frac{1}{5}2^{-5i} - .....]$$

$$= 2^{-i} + 2^{-i-1} + ...$$

$$= 0.00...11...$$

↑↑

ith (i+1)th position,

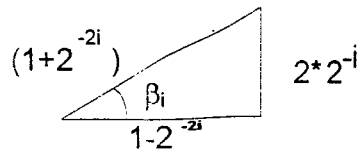
From the above observation, we get corollary 1.

Corollary 1:

Let $\beta_i = 2\tan^{-1} 2^{-i} = 0.\beta_i^1\beta_i^2...\beta_i^i\beta_i^{i+1}...$, we observe that the (i)th and the(i+1)th bits are always 11 respectively, and before the (i)th bit all bits are zeros. Namely, $(\beta_i^i,\beta_i^{i+1}) = (1,1)$,and $\beta_i^k = 0$, *if* $k < i$.

3.1.A.2 The properties of angle β:

Since $\beta_i = 2\tan^{-1} 2^{-i}$, the following holds:



Figuure 3.1.B **The** triangle of β_i

(a) $\cos\beta_i = \dfrac{1-2^{-2i}}{1+2^{-2i}}$

(b) $\tan\beta_i = \dfrac{2\cdot 2^{-i}}{1-2^{-2i}}$

(c)

$$\frac{1}{1+2^{-2i}} = 1-2^{-2i}+2^{-4i}-2^{-6i}+2^{-8i}-\ldots..$$

when i=3,

$$\frac{1}{1+2^{-6}} = 1-2^{-6}+2^{-12}-2^{-18}+2^{-24}+\text{Error},$$

$$|\text{Error}| \le 2^{-29}$$

when i=4,

$$\frac{1}{1+2^{-8}} = 1-2^{-8}+2^{-16}-2^{-24}+Error,$$

$$|\text{Error}| \le 2^{-31}$$

3.1.A.3 The execution of the angle β :

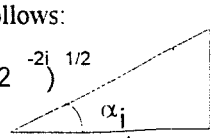
In this algorithm, the rotation angle θ ranges between 0 and $\frac{\pi}{4}$. Now let us examine the execution of the angle β :

$$\text{Let } \theta = z_0 0.z_0^1 z_0^2 \ldots\ldots z_0^n \ldots\ldots z_0 = \begin{cases} 1 \text{ if } \theta < 0 \\ 0 \text{ if } \theta > 0 \end{cases},$$

$$z_0^i \in \{0,1\}$$

$$|\theta| \le \frac{\pi}{4} = 0.1100100\text{1}\ldots..(binary\ represemtation)$$

If the input rotation angle θ is $\frac{\pi}{4}$, we can select angles $\alpha_2, \alpha_3, \alpha_4$ or angles β_3, β_4 to make the first four bits after point of the angle θ converge to zeros. In the above (a) and (c), we use only the five terms in $\cos\beta_3$ and $\cos\beta_4$, and the terms in $\cos\alpha_i$ are as follows:



$$\cos\alpha_i = \frac{1}{\sqrt{1+2^{-2i}}} = (1+2^{-2i})^{-\frac{1}{2}}$$

[By Taylor Series approximation]

$$= 1+\frac{-\frac{1}{2}}{1}2^{-2i}+\frac{(-\frac{1}{2})(-\frac{1}{2}-1)}{1\cdot 2}2^{-4i}$$
$$+\frac{(-\frac{1}{2})(-\frac{1}{2}-1)(-\frac{1}{2}-2)}{1\cdot 2\cdot 3}2^{-6i}+\ldots$$
$$= 1-2^{-2i-1}+3\cdot 2^{-4i-3}-5\cdot 2^{-6i-4}$$
$$+35\cdot 2^{-8i-7}-\ldots.$$

when i=1,

$$\cos\alpha_1 = 1-2^{-3}+2^{-5}-2^{-7}-2^{-8}+2^{-13}+\ldots..$$

i=2,

$$\cos\alpha_2 = 1-2^{-5}+2^{-9}-2^{-11}-2^{-12}-2^{-14}+\ldots..$$

i=3,

$$\cos\alpha_3 = 1-2^{-7}+2^{-13}-2^{-15}-2^{-20}-2^{-22}+\ldots..$$

i=4,

$$\cos\alpha_4 = 1-2^{-9}+2^{-16}-2^{-18}-2^{-26}-2^{-28}+\ldots..$$

We know that the Taylor series approximation of $\cos\alpha_2, \cos\alpha_3$ *and* $\cos\alpha_4$ contains a lot more terms than that in $\cos\beta_3$ and $\cos\beta_4$. It means that if we take the angle α_i instead of β_i for computation before i=5, we must perform more shift and add operations. After i=5, the number of terms in both case are almost equal. So we take β_3 *and* β_4 for input angle computation before i=5, after i=5, we take α_i for computation shown in case 2. We observe that after at most three subtractions, the first four bits of the input angle θ are all zeros, we can use angle α_i to enable the remaining angles converge to zero.

(B)Case 2(The first four bits after the point are all zeros)

3.1.B.1 Calculation of the appropriate values of q_i's for a given θ :

(B1) Contents of the prestoied angle values in the ROM

$$\alpha_i = \tan^{-1}2^{-i} = 2^{-i} - \frac{1}{3}2^{-3i} + \frac{1}{5}2^{-5i} + \ldots.$$
$$= 2^{-i-1}+2^{-i-2}+\ldots..$$

Therefore, α_i can be expressed as

$$\alpha_i = 0.00\ldots11\,\alpha_i^{i-3}\alpha_i^{i-4}\alpha_i^{i-5}\ldots$$
$$\uparrow$$
(i+1)th position ,

$\alpha_i^j \in \{0,1\}$, i=0,1,2.... j=i+3,i+4...

(B2) The choice of (q_i, q_{i+1})

Let $\theta = z_0 = z_0^0 0.z_0^1 z_0^2 \ldots z_0^n \ldots$, where 0 means the integral part in our fixed-point format. $z_0^0=0$ if θ is positive, or $z_0^0=1$ if θ is negative. The following is an explanation of how to choose proper values (q_i, q_{i+1}) considering two bits at each time.

case1 $\theta > 0$ $(z_0^0=0)$

(i) If $z_0^1 z_0^2 = 11$, then we set $q_0 = 1$, $q_1 = 0$

$$z_0 = 00.11 z_0^3 z_0^4 \ldots\ldots$$
$$- \alpha_0 = 00.11 \alpha_0^3 \alpha_0^4 \ldots\ldots$$
$$z_2 = z_2^0 0.00 z_2^3 z_2^4 \ldots\ldots$$

$z_2^i \in \{0,1\}$, i=0,3,4.....

(ii) If $z_0^1 z_0^2 = 10$ or 01, then we set $q_0 = 0$, $q_1 = 1$

$$z_0 = 00.10 z_0^3 z_0^4 z_0^5 \ldots\ldots$$
$$- \alpha_1 = 00.01 1 \alpha_0^4 \alpha_0^5 \ldots\ldots$$
$$z_2 = z_2^0 0.00 z_2^3 z_2^4 \ldots\ldots$$

or

$$z_0 = 00.01 z_0^3 z_0^4 z_0^5 \ldots\ldots$$
$$- \alpha_1 = 00.01 1 \alpha_0^4 \alpha_0^5 \ldots\ldots$$
$$z_2 = z_2^0 0.00 z_2^3 z_2^4 \ldots\ldots$$

(iii) If $z_0^1 z_0^2 = 00$, then we set $q_0 = 0$, $q_1 = 0$

$$z_0 = 00.00 z_0^3 z_0^4 z_0^5 \ldots\ldots$$
$$- \quad 0 = 00.00000 \ldots\ldots$$
$$z_2 = z_2^0 0.00 z_2^3 z_2^4 \ldots\ldots . z_2^i = z_0^i, i = 3,4,5,\ldots\ldots$$

case2 $\theta < 0$ $(z_0^0=1)$

We use similar methods mentioned above to get the proper values (q_i, q_{i+1}).

If we select (q_i, q_{i+1}) according to algorithm (★) given below, we can make the z-variable to converge to zero:

$$(q_i, q_{i+1}) = \begin{cases} (1,0) \\ (0,1) \\ (0,0) \\ (-1,0) \\ (0,-1) \\ (0,0) \end{cases}$$

$$\text{if } (z_i^0 z_i^1 z_i^{i-1}) = \begin{cases} (011) \\ (001) \text{ or } (010) \\ (000) \\ (100) \\ (101) \text{ or } (110) \\ (111) \end{cases} \quad \ldots\ldots\ldots(\star)$$

From the above discussions, we proved corollary 2.

Corollary 2:

According to the n-bit input angle θ, we can decide the proper values (q_i, q_{i+1}) by examining two bits at each time and after $\left\lceil \frac{n}{2} \right\rceil$ computations, we obtain $\left| \theta - \sum_{i=0}^{n-1} q_i \alpha_i \right| < 2^{-n}$.

3.1.B.2 Taylor series approximation of $\cos\alpha_i$

Using Taylor series approximation, $\cos\alpha_i$ can be expressed as follows:

$$\cos\alpha_i = \frac{1}{\sqrt{1+2^{-2i}}} = (1+2^{-2i})^{-\frac{1}{2}}$$
$$= 1 - 2^{-2i-1} + 2^{-4i-1} - 2^{-4i-3} + Error, \ |Error| \le 2^{-39}$$

, for i≧6.

By the above two cases, given vector P(x,y) and rotation angle θ, We can set the initial value x, y and z : $x_0 = x, y_0 = y, z_0 = \theta$, then after computation,

$z_n \approx 0, x_n \approx x\cos\theta - y\sin\theta = x'$,

$y_n \approx x\sin\theta + y\cos\theta = y'$

By the above observations, we get the following algorithm that performs coordinate rotation:

[Input]: x, y, $\theta = z_0^0 0.z_0^1 z_0^2 \ldots\ldots z_0^n \ldots$

(initial) *set* $x_0 \leftarrow A$, $y_0 \leftarrow B$, $z_0 \leftarrow \theta$,

Step 1:

(1) *scan* $\left[z_0^0 z_0^1 z_0^2 \right]$:

$$\begin{cases} \text{if } [z_0^0 z_0^1 z_0^2] = [000] \text{ or } [z_0^0 z_0^1 z_0^2] = [111] \\ \quad \text{then goto step 2} \\ \text{if } [z_0^0 z_0^1 z_0^2] = [001] \text{ or } [z_0^0 z_0^1 z_0^2] = [010] \text{ or } [z_0^0 z_0^1 z_0^2] = [011] \\ \quad \text{then set } P_3 = 1 \text{ goto (2)} \\ \text{if } [z_0^0 z_0^1 z_0^2] = [100] \text{ or } [z_0^0 z_0^1 z_0^2] = [101] \text{ or } [z_0^0 z_0^1 z_0^2] = [110] \\ \quad \text{then set } P_3 = -1 \text{ goto (2)} \end{cases}$$

$$(2)\begin{cases} x_1 = (1 - 2^{-6} + 2^{-12} - 2^{-18} + 2^{-24})(x_0 - 2^{-6} y_0 - P_3 \cdot 2 \cdot 2^{-3} y_0) \\ y_1 = (1 - 2^{-6} + 2^{-12} - 2^{-18} + 2^{-24})(y_0 - 2^{-6} y_0 + P_3 \cdot 2 \cdot 2^{-3} x_0) \\ z_1 = z_0 - P_3 \beta_3 \qquad (\beta_3 = 2 \cdot \tan^{-1} 2^{-3}) \end{cases}$$

(3) *set* $x_0 \leftarrow x_1$, $y_0 \leftarrow y_1$, $z_0 \leftarrow z_1$: *goto*(1)

Step 2:

(4) *scan* $\left[z_0^0 z_0^3 z_0^4 \right]$:

$$\begin{cases} \text{if } [z_0^0 z_0^3 z_0^4] = [000] \text{ or } [z_0^0 z_0^3 z_0^4] = [111] \\ \quad \text{then goto step 2} \\ \text{if } [z_0^0 z_0^3 z_0^4] = [001] \text{ or } [z_0^0 z_0^3 z_0^4] = [010] \text{ or } [z_0^0 z_0^3 z_0^4] = [011] \\ \quad \text{then set } P_4 = 1 \text{ goto (5)} \\ \text{if } [z_0^0 z_0^3 z_0^4] = [100] \text{ or } [z_0^0 z_0^3 z_0^4] = [101] \text{ or } [z_0^0 z_0^3 z_0^4] = [110] \\ \quad \text{then set } P_4 = -1 \text{ goto (5)} \end{cases}$$

$$(5)\begin{cases} x_1 = (1 - 2^{-8} + 2^{-16} - 2^{-24})(x_0 - 2^{-8} y_0 - P_4 \cdot 2 \cdot 2^{-4} y_0) \\ y_1 = (1 - 2^{-8} + 2^{-16} - 2^{-24})(y_0 - 2^{-8} y_0 + P_4 \cdot 2 \cdot 2^{-4} x_0) \\ z_1 = z_0 - P_4 \beta_4 \qquad (\beta_4 = 2 \cdot \tan^{-1} 2^{-4}) \end{cases}$$

(6) set $x_0 \leftarrow x_1$, $y_0 \leftarrow y_1$, $z_0 \leftarrow z_1$: goto(4)

Step3:

(7) set $x_0 \leftarrow x_1, y_0 \leftarrow y_1, z_0 \leftarrow z_1$: goto (4)

(8)

for(i=4; I<23; i=i+2)

{

$$(q_i, q_{i-1}) = \begin{cases} (1,0) \text{ if } [z_i^0 z_i^{i+1} z_i^{i+2}] = [011] \\ (0,1) \text{ if } [z_i^0 z_i^{i+1} z_i^{i+2}] = [001] \text{ or } [010] \\ (0,0) \text{ if } [z_i^0 z_i^{i+1} z_i^{i+2}] = [000] \text{ or } [111] \\ (-1,0) \text{ if } [z_i^0 z_i^{i+1} z_i^{i+2}] = [100] \\ (0,-1) \text{ if } [z_i^0 z_i^{i+1} z_i^{i+2}] = [101] \text{ or } [110] \end{cases}$$

$$Z_{i+2} = Z_i - q_i \alpha_i - q_{i-1} \alpha_{i-1} \cdots (*)$$

(where $\alpha_i = \tan^{-1} 2^{-i}$)

$$X_{i+2} = [1 - |q_i|(2^{-2i-1} - 2^{-4i-1} + 2^{-4i-3})]$$
$$\times [1 - |q_{i+1}|(2^{-2i-3} - 2^{-4i-5} + 2^{-4i-7})]$$
$$\times [X_i + (q_i 2^{-i} + q_{i-1} 2^{-i-1}) Y_i];$$

$$Y_{i+2} = [1 - |q_i|(2^{-2i-1} - 2^{-4i-1} + 2^{-4i-3})]$$
$$\times [1 - |q_{i+1}|(2^{-2i-3} - 2^{-4i-5} + 2^{-4i-7})]$$
$$\times [Y_i - (q_i 2^{-i} + q_{i+1} 2^{-i-1}) X_i];$$

}

[Output]: $x'(= X_n), y'(= Y_n), 0(= Z_n)$

In the above algorithm, step 1 checks the first two bits of the input angle θ (e.g., z_0^1 and z_0^2). If they equal zero, the algorithm jumps to step2 and checks the third and the fourth bits of the input angle θ (e.g., z_0^3 and z_0^4) . If not, the algorithm sets the operand value P_3 according to the value of $[z_0^0 z_0^1 z_0^2]$, performs the operations (2) and (3). We repeat the same process until z_0^1 and z_0^2 are both zero . In step2, we check the third and the fourth bits of the input angle θ (e.g., z_0^3 and z_0^4), if they do not equal zero then the algorithm sets the operand value P_3 according to the value of $[z_0^0 z_0^3 z_0^4]$, performs the operations (5) and (6). This will make z_0^3 and z_0^4 become 00, then the algorithm performs step3. In step3, we set the initial value of i to be four because the first four bits of angle θ are all zeros, there is no XY_ROM access execution in this algorithm because the initial value x_0 and y_0 is set in the initial execution. After the execution of this algorithm, the output value $(x_n, y_n) = (x', y')$ is the new coordinate value of (x,y). It is obvious that no multiplications or divisions are involved in our new algorithm. Thus the execution is faster than the algorithm which uses constant factor. If we want to compute the length of the two edges x and y in an right triangle using slides ρ and angle θ (see Figure 3.2), we set the initial value $z_0 = \theta, x_0 = \rho, y_0 = 0$. then we can obtain x and y.

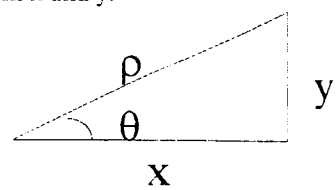


Figure 3.2 Right triangle

3.2 Simulation Results

We simulate the algorithm by C language on the SUN Sparc-10 workstation, then we

examine the correctness by subtracting the value which the algorithm computed from the C standard library. We have the following Figures to explain our algorithm accuracy. The value E equals the difference between the computed value(e.g. by the algorithm) and the theoretical value(e.g. by the standard C library).

3.2.1 The error analysis computed by our algorithm

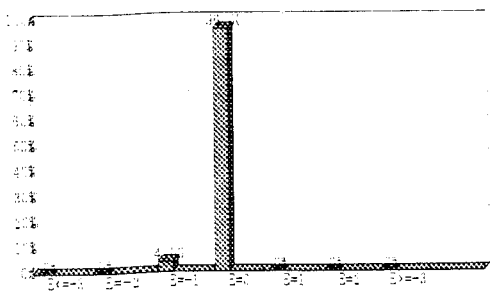


Figure 3.3 The erro r analysis of Xn

In Figure 3.3. we find that the Xn can achieve 96% correct rate. there is only 4.0% error rate.

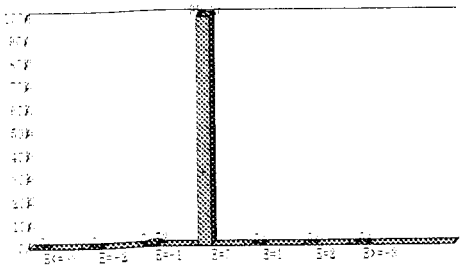


Figure 3.4 The error analysis of Yn

In Figure 3.4. we find that the Yn can achieve 99.28% correct rate. there is only 0.72% error rate. From these simulation results. we find that the algorithm only takes at most 12 iterations to complete the computation for performing 23-bit data format computation. It is almost half the number of iterations when using classic CORDIC methods. We have simulated the classical CORDIC algorithm using the same data formats. we get the following results.

3.2.2 The error analysis computed by classical CORDIC algorithm

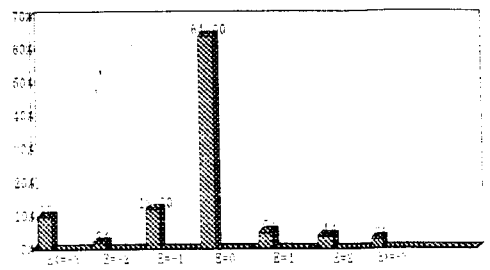


Figure 3.5 The error analysis of Xn computed by classic CORDIC algorithm

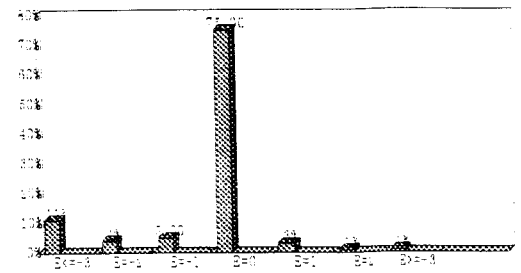


Figure 3.6 The error analysis of Yn computed by classic CORDIC algorithm

From the above we can understand that the classic CORDIC algorithm can only achieve 75% correct rate. Besides. for 23-bit data formats. the CORDIC algorithm needs 23 iterations and one multiplication of the final value by a constant factor. thus reduce the correctness and the execution performance.

3.3 Speedup comparisons

In above algorithms we derive. the iterations for n-bit data is 0.5n. when compared to the other proposed papers. we find the accuracy of our algorithm is better than others. Table 3.1 is our iteration comparisons.

Table 3.1 Iteration comparisons of proposed algorithms

Methods	Iterations of algorithm
The classical CORDIC[8]	1.5n(including the multiplication time)
Lin and Sips[5]	n+6
Takagi *et al.*[7]	n
Lee and Lang[4]	1.125n
Timmermann *et al.*[10]	(9n-3)/8
Our new algorithm	*0.5*n

4. Rectangular Coordinate to Polar Coordinate Transformation Using New CORDIC Algorithm

In this section we will define a new CORDIC methods that can perform the transformation from rectangular coordinate (x,y) to polar coordinate (ρ,θ) and vice versa. The transformation plays an important role in the computer graphics. In part 4.1, we describe our derivation for our new algorithm, and in part 4.2 we lists our simulation results.

Consider the rectangular coordinate (x,y) and polar coordinate (ρ,θ) in the plane shown in Figure 4.1, we have the following relations between x, y, ρ and θ.

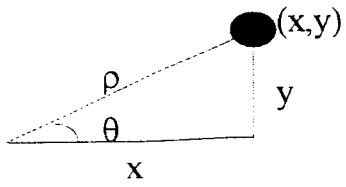


Figure 4.1 The rectangular coordinate and polar coordinate

$$\begin{cases} x = \rho\cos\theta \\ y = \rho\sin\theta \end{cases} \leftrightarrow \begin{cases} \rho = \sqrt{x^2 + y^2} \\ \theta = \tan^{-1}\dfrac{y}{x} \end{cases} \quad (4\text{-}1)$$

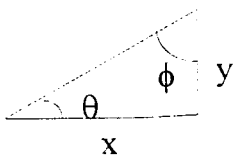
4.1 Derivation Process

4.1.1 Rectangular coordinate to polar coordinate transformation($(x,y) \rightarrow (\rho,\theta)$)

Now, given rectangular coordinate (x,y), we want to find the polar coordinate (ρ,θ), where $\rho = \sqrt{x^2 + y^2}$, $\theta = \tan^{-1}\dfrac{y}{x}$. We consider two cases: (1)$x \geq y > 0$; (2)$y > x > 0$.

(1) $x \geq y > 0$

If $x \geq y > 0$, then $\dfrac{\pi}{4} \geq \theta = \tan^{-1}\dfrac{y}{x} > 0$.



for arbitrary angle ϕ.

(i) if $\dfrac{\pi}{2} > \phi > \theta$ $\quad \sin\theta = \dfrac{y}{\sqrt{x^2 + y^2}}$

$$y\cos\phi < y\cos\theta = x\sin\theta < x\sin\phi$$

$$\cos\theta = \frac{x}{\sqrt{x^2 + y^2}}$$

(ii) if $\theta > \phi > 0$

$$y\cos\phi > y\cos\theta = x\sin\theta > x\sin\phi$$

By (i) and (ii), for $0 < \phi < \dfrac{\pi}{2}$.

$$\begin{cases} \phi > \theta \Leftrightarrow y\cos\phi - x\sin\phi < 0 \\ \phi \leq \theta \Leftrightarrow y\cos\phi - x\sin\phi \geq 0 \end{cases}$$

We get our algorithm as follows:

$$\begin{cases} X_{i-1} = (X_i - q_i 2^{-k_i} Y_i)\cos\alpha_{k_i} \\ Y_{i-1} = (Y_i + q_i 2^{-k_i} X_i)\cos\alpha_{k_i} \\ Z_{i-1} = Z_i + q_i\alpha_{k_i} \\ K_i = \exp(Y_i) - \exp(X_i) \end{cases}$$

where $\alpha_{k_i} = \tan^{-1} 2^{-k_i}$ and $q_i = \begin{cases} +1, & \text{if } X_i > 0 \\ -1, & \text{if } X_i < 0 \end{cases}$

the initial value of X, Y, Z, and K is $X_0 = y$, $Y_0 = x$, $Z_0 = 0$, and $K_0 = \exp(Y_0) - \exp(X_0)$. The function **exp**(N) is the exponent of number *N*.

Here is the proof of our new algorithm.

(proof)

Setting the initial value $X_0 = y$, $Y_0 = x$, $Z_0 = 0$, and $K_0 = \exp(Y_0) - \exp(X_0)$.

After the first iteration,

$$X_1 = (y - q_0 2^{-k_0} x)\cos\alpha_{k_0}$$
$$= y\cos\alpha_{k_0} - x\sin q_0\alpha_0$$

$$Y_1 = (x + q_0 2^{-k_0} y)\cos\alpha_{k_0}$$
$$= x\cos\alpha_{k_0} + y\sin q_0\alpha_0$$

$$Z_1 = q_0\alpha_{k_0}$$

$$K_1 = \exp(Y_1) - \exp(X_1)$$

After the second iteration,

$$X_2 = (X_1 - q_1 2^{-k_1} Y_1)\cos\alpha_{k_1}$$
$$= (y\cos\alpha_{k_0}\cos\alpha_{k_1} - x\sin q_0\alpha_{k_0}\cos\alpha_{k_1})$$
$$-q_1(x\cos\alpha_{k_0}\sin\alpha_{k_1} + y\sin q_0\alpha_{k_0}\sin\alpha_{k_1})$$
$$= y(\cos\alpha_{k_0}\cos\alpha_{k_1} - \sin q_0\alpha_{k_0}\sin q_1\alpha_{k_1})$$
$$-x(\sin q_0\alpha_{k_0}\cos\alpha_{k_1} + \cos\alpha_{k_0}\sin q_1\alpha_{k_1})$$
$$= y\cos(q_0\alpha_{k_0} + q_1\alpha_{k_1}) - x\sin(q_0\alpha_{k_0} + q_1\alpha_{k_1})$$

$$Y_2 = x\cos(q_0\alpha_{k_0} + q_1\alpha_{k_1}) + y\sin(q_0\alpha_{k_0} + q_1\alpha_{k_1})$$

$$Z_2 = q_0\alpha_{k_0} + q_1\alpha_{k_1}.$$

$$K_2 = \exp(Y_2) - \exp(X_2)$$

After n iterations, we can obtain that

$$\begin{cases} X_n = y\cos(\sum_{i=0}^{n-1} q_i\alpha_{k_i}) - x\sin(\sum_{i=0}^{n-1} q_i\alpha_{k_i}) \\ Y_n = x\cos(\sum_{i=0}^{n-1} q_i\alpha_{k_i}) + y\sin(\sum_{i=0}^{n-1} q_i\alpha_{k_i}) \\ Z_n = \sum_{i=0}^{n-1} q_i\alpha_{k_i} \\ K_n = \exp(Y_n) - \exp(X_n) \end{cases}$$

since $Y_0 = x > X_0 = y, K_0 = \exp(x) - \exp(y)$.

so

$$\exp(X_1) = \exp((y - 2^{-k_0}x)\cos\alpha_{k_0}) < \exp(y) - 1$$

By the same way.

$$\exp(X_2) = \exp((X_1 - 2^{-k_1}Y_1)\cos\alpha_{k_1}) < \exp(X_1) - 1 < \exp(y) - 2$$

After n iterations, we have

$$\exp(X_n) < \exp(X_{n-1}) - 1 < \ldots < \exp(X_1) - (n-1) < \exp(X_0) - n = \exp(y) - n$$

At most n=exp(y)+24 iterations, we obtain $\exp(X_n) < -24$. Thus we can conclude that when X_n converges to zero, we have the following results:

$X_n \to 0$, ($|X_n| < 2^{-24}$)

$$\Rightarrow y\cos(\sum_{i=0}^{n-1} q_i\alpha_i) = x\sin(\sum_{i=0}^{n-1} q_i\alpha_i)$$

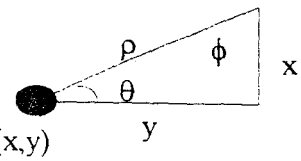
$$\tan(\sum_{i=0}^{n-1} q_i\alpha_i) = \frac{y}{x},$$

and $Z_n = \sum_{i=0}^{n-1} q_i\alpha_i \Rightarrow \theta = \tan^{-1}\frac{y}{x}$

$$Y_n = x\cos\theta + y\sin\theta = x\cdot\frac{x}{\sqrt{x^2+y^2}} + y\cdot\frac{y}{\sqrt{x^2+y^2}} = \frac{x^2+y^2}{\sqrt{x^2+y^2}} = \sqrt{x^2+y^2} = \rho$$

(2)y>x>0

If y>x>0, then we let the initials value of X, Y, Z, and K are $X_0 = x$, $Y_0 = y$, $Z_0 = 0$, and $K_0 = \exp(Y_0) - \exp(X_0)$, and $\theta = \frac{\pi}{2} - \phi$, then apply them into algorithm, we can get the polar coordinate (ρ,θ).



From the above derivations, we obtain the following algorithm:

[Input] $X_0 = y, Y_0 = x, Z_0 = 0$

[Initial] $K_0 = \exp(Y_0) - \exp(X_0)$

for(i=0;i<=23;i++){

Step1: $0 \le K_i \le 3$

$$\begin{cases} X_{i+1} = (X_i - 2^{-6}X_i - q_i 2\cdot 2^{-3}Y_i) \times (1 - 2^{-6} + 2^{-12} - 2^{-18} + 2^{-24}) \\ Y_{i+1} = (Y_i - 2^{-6}Y_i + q_i 2\cdot 2^{-3}X_i) \times (1 - 2^{-6} + 2^{-12} - 2^{-18} + 2^{-24}) \\ Z_{i+1} = Z_i + q_i\alpha_{k_i} \\ K_{i+1} = \exp(Y_{i+1}) - \exp(X_{i+1}) \end{cases}$$

Step2: $4 \le K_i \le 6$

$$\begin{cases} X_{i+1} = (X_i - 2\cdot 2^{-2k_i}X_i - q_i 2\cdot 2^{-k_i}Y_i) \times (1 - 2^{-2k_i} + 2^{-4k_i} - 2^{-6k_i}) \\ Y_{i+1} = (Y_i - 2\cdot 2^{-2k_i}Y_i + q_i 2\cdot 2^{-k_i}X_i) \times (1 - 2^{-2k_i} + 2^{-4k_i} - 2^{-6k_i}) \\ Z_{i+1} = Z_i + q_i\alpha_{k_i} \\ K_{i+1} = \exp(Y_{i+1}) - \exp(X_{i+1}) \end{cases}$$

Step3: $K_i \ge 7$

$$\begin{cases} X_{i+1} = (X_i - q_i\cdot 2^{-k_i}Y_i) \times (1 - 2^{-2k-1}) \\ Y_{i+1} = (Y_i + q_i\cdot 2^{-k_i}X_i) \times (1 - 2^{-2k-1}) \\ Z_{i+1} = Z_i + q_i\alpha_k \\ K_{i+1} = \exp(Y_{i+1}) - \exp(X_{i+1}) \end{cases}$$

}

where $q_i = \begin{cases} +1, & \text{if } X_i \ge 0 \\ -1, & \text{if } X_i < 0 \end{cases}$

[Output] $X_n \to 0, Y_n = \rho, Z_n = \theta$

4.1.2 Polar coordinate to rectangular coordinate transformation($(\rho,\theta) \to (x,y)$)

If we want to transform the polar coordinate to rectangular coordinate, we can make use of the algorithm mentioned in Section 3, setting the initial value $z_0 = \theta, x_0 = \rho, y_0 = 0$, then we can obtain x and y.

4.2 Simulation Results

We simulate these two algorithms by C language program to check their accuracy for 131,172 input patterns totally. In 4.2.1 we describe the simulation results of the transformation from rectangular coordinate to

polar coordinate. In 4.2.5 we describe the simulation results of the transformation of polar coordinate to rectangular coordinate. The symbol E denotes the difference between the computed value and the standard value.

4.2.1 Error analysis of rectangular coordinate to polar coordinate transformation($(x,y) \rightarrow (\rho,\theta)$)

4.2.1a Error analysis of ρ computed by our algorithm

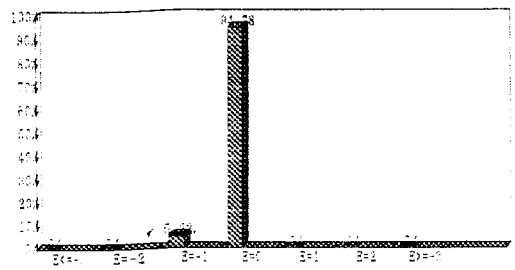


Figure 4.2 The error analysis of 5.2.1a

In Figure 4.2 we find that the accuracy of ρ computed by our algorithm is 94.78%. there is only 5.22% one-bit error.

4.2.1b Error analysis of θ computed by our algorithm

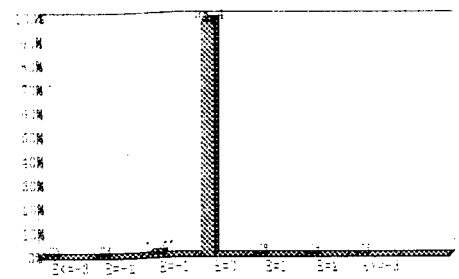


Figure 4.3 The error analysis of 4.2.1b

In Figure 4.3 we find that the accuracy of θ computed by our algorithm is 98.43%. there is only 1.57% one-bit error.

4.2.2 Error analysis of rectangular coordinate to polar coordinate transformation computed by classical CORDIC

4.2.2a Error analysis of ρ computed by classical CORDIC

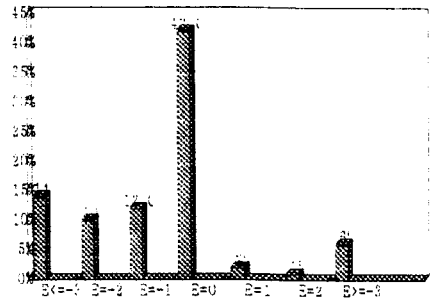


Figure 4.4 The error analysis of 4.2.2a

In above results we find that the accuracy computed by classical CORDIC only achieves 42% accuracy.

4.2.2b Error analysis of θ computed by classical CORDIC

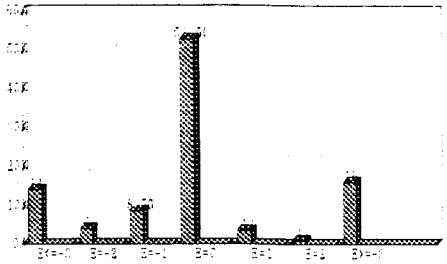


Figure 4.5 The error analysis of 4.2.2b

In above results we find that the accuracy computed by classical CORDIC only achieves 52.5% accuracy.

4.2.3 Error analysis of polar coordinate to rectangular coordinate transformation($(\rho,\theta) \rightarrow (x,y)$)

4.2.3a Error analysis of x computed by our algorithm

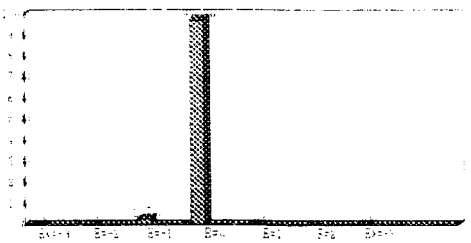


Figure 4.6 The error analysis of 4.2.3a

In Figure 4.6 we find that the accuracy of x computed by our algorithm is 97.3%. there is only 2.7% one-bit error.

4.2.3b Error analysis of y computed by our algorithm

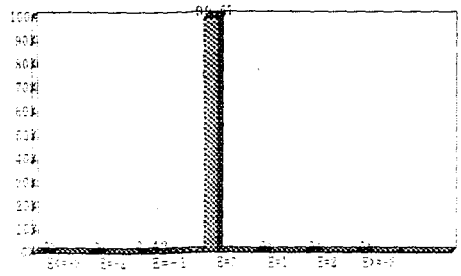


Figure 4.7 The error analysis of 4.2.3b

In Figure 4.7 we find that the accuracy of y computed by our algorithm is 99.57%, there is only 0.43% one-bit error.

4.2.4 Error analysis of polar coordinate to rectangular coordinate transformation computed by classical CORDIC

4.2.4a Error analysis of x computed by classical CORDIC

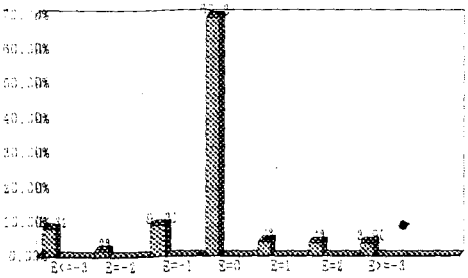


Figure 4.8 The error analysis of 4.2.4a

In above results we find that the accuracy computed by classical CORDIC only achieves 69.37% accuracy.

4.2.4b Error analysis of y computed by classical CORDIC

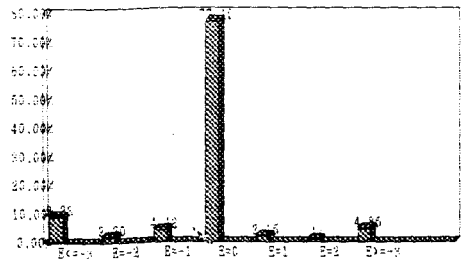


Figure 4.9 The error analysis of 4.2.4b

In above results we find that the accuracy computed by classical CORDIC only achieves 77.15% accuracy. In the above simulation results, we find that the accuracy of our new algorithm is very high. We also find that the maximum number of iterations for our new algorithm is 14.68 almost equal to 15 which is less than the number of iterations needed when using the classical CORDIC.

5. Conclusion

In this paper, we make use of variable factor to develop an algorithm that can perform plane rotation between two vectors in the plane. The advantage of using variable factor is that it is multiplication-free and division free and thus it reduces execution time and increases accuracy. Besides, we keep the original concept (e.g., shift-and-add) of classical CORDIC algorithm in our new design.

In future research, one can use the same method to develop new CORDIC algorithms which applied in the floating-point formats.

6. References

[1]J. A. Lee, T. Lang, "Constant-Factor Redundant CORDIC for Angle Calculation and Rotation," *IEEE Trans. Comput.*, vol. 41, no. 8, pp.1016-1025, Aug. 1992.

[2]H. X. Lin, and H. J. Sips, "On-line CORDIC Algorithms," IEEE Trans. Comput., vol. 39, no. 8, pp.1038-1052, Aug. 1990.

[3]N. Takagi, T. Asads, and S. Yajima, "Redundant CORDIC Methods with a Constant Scalar Factor for Sine and Cosine Computation," IEEE Trans. Comput., vol. 40, no. 9, pp.989-995, Sept. 1991.

[4]J. E. Volder, "The CORDIC Trigonometric Computing Technique," IRE Trans. Electron. Comput., vol. EC-8, no. 3, pp. 330-334, Sept. 1959.

[5]B. Yang, D. Timmermann, J. F. Bohme, H. Hahn, B. J. Hosticka, G. Schmidt, and G. Zimmer, "Special Computers: Graphics, Robotics," in Proc. VLSI Comput., Compeuro, pp. 727-730, May 1987.

[6]K. S. Yang, "A New Method of Implementation of VLSI CORDIC for Sine and Cosine Computation," *Master Thesis*, Fen-Chia University, January, 1984.