CORDIC
Ivor Page[1]

15.1 Summary

The CORDIC (COrdinate Rotation DIgital Computer) algorithms provide a unified way to generate elementary functions, such as multiplication, division, sin, cos, tan, arctan, ln, exp, square root, sinh, cosh, and tanh, using only shifts and additions of fixed point quantities. Therefore a simple integer adder and shifter are all the hardware needed. The algorithms produce results one digit at a time. Faster convergence techniques are available for high speed operation where floating point hardware is available. Truncated power series are used in digital computers for function evaluation. Approximations to these functions also use ratios of two polynomials, a method invented by Cecil Hastings, and known as Hastings Approximations.

The CORDIC algorithms are based on simple coordinate rotations. See figure 1 below, where the vector is rotated through α radians:

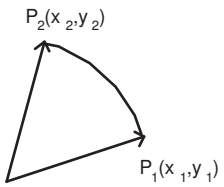

Figure 1: CORDIC Rotation

Calculation of the second endpoint coordinates is simple:

$$\begin{aligned} x_2 &= x_1 \cos\alpha - y_1 \sin\alpha \\ y_2 &= y_1 \cos\alpha + x_1 \sin\alpha \end{aligned}$$

Completing this calculation in one step requires knowledge of $\cos\alpha$ and $\sin\alpha$.

[1]University of Texas at Dallas

Instead of the rotation in figure 1, we do a pseudo rotation through α radians as shown below:

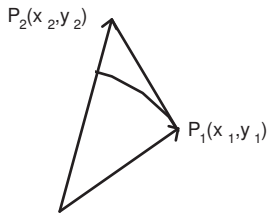

Figure 2: CORDIC Pseudo Rotation

The equations now become:

$$\begin{aligned} x_2 &= x_1 - y_1 \, tan \, \alpha \\ y_2 &= y_1 + x_1 \, tan \, \alpha \end{aligned}$$

The length of the vector grows by the factor $\sqrt{1 + tan^2 \alpha}$. To see this, consider:

$$\begin{aligned} r_2^2 &= (x_1 - y_1 \, tan \, \alpha)^2 + (y_1 + x_1 \, tan \, \alpha)^2 \\ &= (x_1^2 + y_1^2)(1 + tan^2 \, \alpha) \\ &= r_1^2(1 + tan^2 \, \alpha) \\ r_2 &= r_1 \sqrt{1 + tan^2 \, \alpha} \end{aligned}$$

We are going to compute the rotation iteratively, so we substitute α_i for α and form the three equations:

$$\begin{aligned} x_{i+1} &= x_i - y_i \, tan \, \alpha_i \\ y_{i+1} &= y_i + x_i \, tan \, \alpha_i \\ z_{i+1} &= z_i - \alpha_i \end{aligned}$$

After m rotations, the radius has grown by the factor:

$$\prod_i \sqrt{1 + tan^2 \, \alpha_i}$$

$$
\begin{aligned}
x_1 &= x_0 - y_0 \, tan \, \alpha_0 \\
y_1 &= y_0 + x_0 \, tan \, \alpha_0
\end{aligned}
$$

$$
\begin{aligned}
x_2 &= x_0(1 - tan \, \alpha_0 \, tan \, \alpha_1) - y_0(tan \, \alpha_0 + tan \, \alpha_1) \\
&= \frac{x_0(cos \, \alpha_0 \, cos \, \alpha_1 - sin \, \alpha_0 \, sin \, \alpha_1) - y_0(sin \, \alpha_0 \, cos \, \alpha_1 + cos \, \alpha_0 \, sin \, \alpha_1)}{cos \, \alpha_0 \, cos \, \alpha_1} \\
&= \frac{x_0 \, cos(\alpha_0 + \alpha_1) - y_0 \, sin(\alpha_0 + \alpha_1)}{cos \, \alpha_0 \, cos \, \alpha_1} \\
y_2 &= y_0(1 - tan \, \alpha_0 \, tan \, \alpha_1) + x_0(tan \, \alpha_0 + tan \, \alpha_1) \\
&= \frac{y_0(cos \, \alpha_0 \, cos \, \alpha_1 - sin \, \alpha_0 \, sin \, \alpha_1) + x_0(sin \, \alpha_0 \, cos \, \alpha_1 + cos \, \alpha_0 \, sin \, \alpha_1)}{cos \, \alpha_0 \, cos \, \alpha_1} \\
&= \frac{y_0 \, cos(\alpha_0 + \alpha_1) + x_0 \, sin(\alpha_0 + \alpha_1)}{cos \, \alpha_0 \, cos \, \alpha_1} \\
&\cdot \\
&\cdot \\
x_m &= k(x_0 \, cos(\textstyle\sum \alpha_i) - y_0 \, sin(\sum \alpha_i)) \\
y_m &= k(y_0 \, cos(\textstyle\sum \alpha_i) + x_0 \, sin(\sum \alpha_i)) \\
z_m &= z_0 - (\textstyle\sum \alpha_i)
\end{aligned}
$$

$$
k = \frac{1}{\prod cos \, \alpha_i} = \prod \sqrt{1 + tan^2 \, \alpha_i}
$$

substitute $\theta = \sum \alpha_i$

$$
\begin{aligned}
x_m &= k[x_0 \, cos \, \theta - y_0 \, sin \, \theta] \\
y_m &= k[y_0 \, cos \, \theta + x_0 \, sin \, \theta] \\
z_m &= z_0 - \theta
\end{aligned}
$$

We can only compute these values in one step if we have a complete table of values of $tan \, \theta$ for every value of θ. Instead we perform the pseudo rotation through the angle θ in iterative steps.

We make $tan \, \alpha_i = d_i 2^{-i}$, $i = 0, 1, 2, \cdots, m$, $d_i \in \{-1, 1\}$, $\alpha_0 = 45^o$, $\alpha_1 = 26.565^o$, and so on.

We always use the same sequence of rotations, 45^o, 26.6^o, 14.0^o, 7.1^o, and so on. To rotate 30 degrees, we choose signs for d_i to make the sum of the angles converge to 30:

$$30.0 \approx 45 - 26.6 + 14.0 - 7.1 + 3.6 + 1.8 - 0.9 + 0.4 \cdots$$

When making z converge to zero, we adjust the sign, d_i, at each iteration to be the same as

the sign of z_i.

The diagram below shows the first three rotations. The initial vector ending at P_1 makes an angle of 30^o with the x axis. The first rotation is -45^o, the second is +26.6^o and the third is -14^o. The iterations end after m rotations.

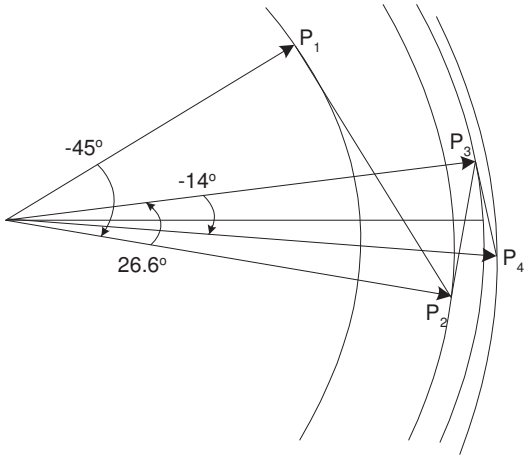

Figure 3: A Sequence of Pseudo Rotations

By sticking to the same sequence of values for α_i, the value of k becomes a constant:

$$k = \prod \sqrt{1 + tan^2\ \alpha_i} = 1.646760258121$$

We store the constant $1/k = 0.607252935$.

In Rotation mode, we iterate to make $z_i \rightarrow 0$ then:

$$\begin{aligned} x_m &= k[x\ cos\ z - y\ sin\ z] \\ y_m &= k[y\ cos\ z + x\ sin\ z] \\ z_m &= 0 \end{aligned}$$

In Vectoring mode, we make $y_i \rightarrow 0$ then:

$$\begin{aligned} x_m &= k\sqrt{x^2 + y^2} \\ y_m &= 0 \\ z_m &= z + tan^{-1}(y/x) \end{aligned}$$

The hardware required is shown below:

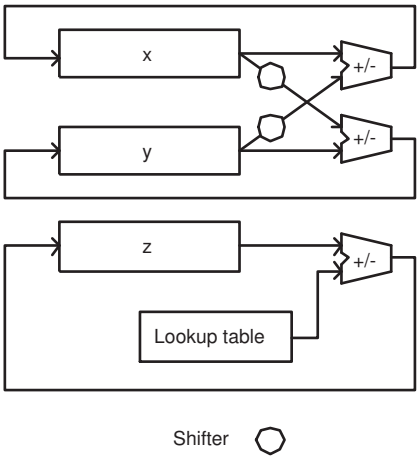

Figure 4: Hardware

Three registers (not necessarily bit serial), two shifters, and three adders are needed. The table stores values of $tan^{-1}(2^{-i})$ for $i = 0, 1, 2, \cdots, m$. The number of bits of precision is given by m.

i	$tan^{-1}(2^{-i})$ in degrees
0	45.0
1	26.6
2	14.0
3	7.1
4	3.6
5	1.8
6	0.9
7	0.4
8	0.2
9	0.1

15.2 Generalized CORDIC

The generalized iterants become:

$$\begin{aligned} x_{i+1} &= x_i - \mu\, d_i\, y_i\, 2^{-i} \\ y_{i+1} &= y_i + d_i\, x_i\, 2^{-i} \\ z_{i+1} &= z_i - d_i\, e_i \end{aligned}$$

The parameter μ has three settings:

$\mu = 1$	Circular Rotation	$e_i = tan^{-1}(2^{-i})$
$\mu = 0$	Linear Rotations	$e_i = 2^{-i}$
$\mu = -1$	Hyperbolic Rotations	$e_i = tanh^{-1}(2^{-i})$

For each setting of μ, two mode are obtained, one where z is made to converge to zero, and the other where y is made to converge to zero. These two modes are known as Rotation Mode and Vectoring Mode, respectively. In Rotation Mode, d_i is set to the sign of z_i. In Vectoring Mode, z_i is set to minus the sign of $x_i\, y_i$ (We don't compute this product, we just get its sign).

We have already seen Rotation and Vectoring mode for circular rotations,

$$\begin{aligned} x_m &= k(x\, cos\, z - y\, sin\, z) \quad [CircularRotationMode] \\ y_m &= k(x\, cos\, z + y\, sin\, z) \\ z_m &= 0 \\ \text{Rule} &: \text{Choose } d_i \in \{-1, 1\} \text{ such that } z \to 0 \end{aligned}$$

$$\begin{aligned} x_m &= k\sqrt{x^2 + y^2} \quad [CircularVectoringMode] \\ y_m &= 0 \\ z_m &= z + tan^{-1}(y/x) \\ \text{Rule} &: \text{Choose } d_i \in \{-1, 1\} \text{ such that } y \to 0 \end{aligned}$$

By setting x_0 or y_0 to $1/k$, sin and cos functions are obtained.

Two new modes are introduced by setting $\mu = 0$. Multiplication and division are provided by these modes. In these rotations, all the intermediate and final points are on the vertical line $x = x_0$.

$$
\begin{aligned}
x_m &= x \quad [LinearRotationMode] \\
y_m &= y + xz \\
z_m &= 0 \\
\text{Rule} &: \text{Choose } d_i \in \{-1, 1\} \text{ such that } z \to 0
\end{aligned}
$$

$$
\begin{aligned}
x_m &= x \quad [LinearVectoringMode] \\
y_m &= 0 \\
z_m &= z + y/x \\
\text{Rule} &: \text{Choose } d_i \in \{-1, 1\} \text{ such that } y \to 0
\end{aligned}
$$

Finally, if $\mu = -1$, two hyperbolic modes are introduced. The iterations are intended to visit points on a hyperbola, but pseudo rotations are used. See the figure below for hyperbolic rotations:

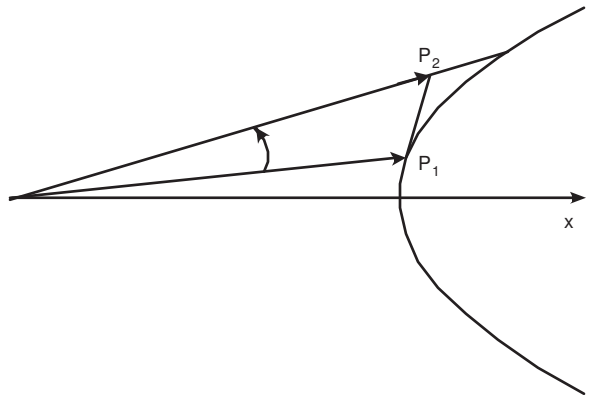

Figure 5: Hyperbolic Pseudo Rotation

The vector length 'expands' by the factor:

$$\sqrt{1 - tanh^2\, \alpha_i} = \frac{1}{cosh\, \alpha_i}$$

Since $cosh\, \alpha_i > 1$ the vector length shrinks. Again, a fixed sequence of values is used for α_i so that the product of these shrinkage factors is a constant. The overall shrinkage factor is $k' = 0.8281593609601$.

Here are the results:

$$\begin{aligned} x_m &= k'(x\,cosh\,z + y\,sinh\,z) \quad [HyperbolicRotationMode] \\ y_m &= k'(y\,cosh\,z + x\,sinh\,z) \\ z_m &= 0 \\ \text{Rule} &: \text{Choose } d_i \in \{-1, 1\} \text{ such that } z \to 0 \end{aligned}$$

$$\begin{aligned} x_m &= k'\sqrt{x^2 - y^2} \quad [HyperbolicVectoringMode] \\ y_m &= 0 \\ z_m &= z + tanh^{-1}(y/x) \\ \text{Rule} &: \text{Choose } d_i \in \{-1, 1\} \text{ such that } y \to 0 \end{aligned}$$

By setting x_0 or y_0 equal to $1/k'$, sinh and cosh functions are obtained.

15.3 Convergence

For circular rotation mode, convergence is guaranteed for $-99.7^o \leq z \leq 99.7^o$, where 99.7 is the sum of all the angles obtained by setting $\alpha_i = tan^{-1}(2^{-i})$. This range includes the range $[-\pi/2, \pi/2]$ in radians. We can extend the range in the usual way:

$$\begin{aligned} cos(z \pm 2j\pi) &= cos\,z \quad j = 0, 1, 2, \cdots \\ sin(z \pm 2j\pi) &= sin\,z \\ cos(z - \pi) &= -cos\,z \\ sin(z - \pi) &= -sin\,z \end{aligned}$$

The Circular Vectoring mode always converges, but it is desirable to limit the range of the fixed point values used in the calculations, so the following identity is used:

$$tan^{-1}(1/y) = \pi/2 - tan^{-1}y$$

Convergence of the hyperbolic modes is only guaranteed if the following iterations are repeated: $4, 13, 40, 121, \cdots, j, 3j + 1, \cdots$. The value of k' computed above already includes allowance for these extra steps.

The extra steps are needed because $tanh^{-1}(2^{-(i+1)}) \geq 0.5 tanh^{-1}(2^{-i})$ doesn't hold in general. With these extra steps, convergence of the hyperbolic modes is guaranteed for $|z| < 1.13$ when computing *sinh* and *cosh*, and for $|y| < 0.81$ when computing *tanh*.

The following identities are used to extend the ranges of the hyperbolic modes:

$$\begin{aligned} cosh(q\,ln\,2 + z) &= 2^{q-1}[cosh\,z + sinh\,z + 2^{-2q}(cosh\,z - sinh\,z)] \\ sinh(q\,ln\,2 + z) &= 2^{q-1}[cosh\,z + sinh\,z - 2^{-2q}(cosh\,z - sinh\,z)] \\ tanh(1 - 2^{-e}s) &= tanh^{-1}(\frac{2 - s - 2^{-e}s}{2 + s - 2^{-e}s}) + \frac{e\,ln\,2}{2} \end{aligned}$$

15.4 The Algorithm

Here is a complete C++ program that implements the CORDIC algorithms:

```
#include<iostream.h>
#include<math.h>
/*                General CORDIC algorithm.
  M=1  gives circular (rotation) mode (sin, cos, sqrt, arctan)
  M=0  gives straight line case (multiply, divide)
  M=-1 gives hyperbolic case (sinh, cosh, sqrt, arctanh)
  D=0  forces z to zero
  D=1  forces angle to zero
*/

void main() {
    double x,y,z;
    double table[54], table2[54];
    double p;
    int D, M;
    cout << endl << "Input D, M, x, y, z: ";
    cin >> D >> M >> x >> y >> z;
    double xx, yy;

    cout.precision(8);
    cout << endl << "   i       2^(-i)";
    cout << "             x               y               z" << endl;

    if(M<=0) p = 0.5;
    else p = 1.0;
    double p2 = p;
    for(int i=0;i<54;i++) {      // initialize arrays
        if(M>0) {
            table[i] = atan(p2);  // radians version
```

```
            p2 /= 2.0;
        }
        if(M<0) {
            table2[i] = 0.5*log((1+p2)/(1-p2));
            if(i!=3 && i!=12 && i!= 39)
                p2 /=2.0;
        }
    }

    for(i=0;i<54;i++) {                        // CORDIC Iterations
        if(D==1 && y<0 || D==0 && z>=0) { // force y or z to zero
            xx = x - M*p*y;
            yy = y + p*x;
            if(M==0)
                z -= p;
            else if(M>0)
                z -= table[i];    // arctan(2^{-i})
            else
                z -= table2[i];    // arctanh(2^{-i})
        }
        else {
            xx = x + M*p*y;
            yy = y - p*x;
            if(M==0)
                z +=p;
            else if(M>0)
                z += table[i];  // arctan(2^{-i})
            else
                z += table2[i]; // arctanh(2^{-i})
            }
        cout << endl;
        cout.width(4);   cout << i << "\t";
        cout.width(12);  cout << p << "\t";
        cout.width(12);  cout << x << "\t";
        cout.width(12);  cout << y << "\t";
        cout.width(12);  cout << z;
        x = xx;
        y = yy;
        if(M>=0 || i!=3 && i!=12 && i!=39)
            p /= 2.0;
    }
    cout << endl;
}
```