

CORDIC algorithm

The **CORDIC (COordinate Rotation DIgital Computer) algorithm** is an efficient iterative method used for computing a variety of mathematical functions, including trigonometric, hyperbolic, logarithmic, and exponential functions. It is particularly useful for hardware implementations where multiplication and division are expensive operations since it relies only on shifts and additions.

Key Features of the CORDIC Algorithm

- **No multiplications or divisions:** Only addition, subtraction, and bit shifts.
- **Efficient for hardware:** Used in FPGA, DSP, and embedded systems.
- **Iterative approach:** Each iteration refines the approximation.

How CORDIC Works

CORDIC operates in two main modes:

1. **Rotation Mode:** Used to compute sine and cosine.
2. **Vectoring Mode:** Used to compute magnitude and phase.

It works by iteratively rotating a vector in a 2D plane using predefined angle values and adjusting the angle until it converges to the desired result.



Applications of CORDIC

- Computing trigonometric functions (sin, cos, tan)
- Vector magnitude computation (hypotenuse)
- Angle computation (atan, atan2)
- Exponential and logarithm calculations
- Complex number operations (e.g., FFT)
- 3D graphics and signal processing
- GPS and SDR (Software Defined Radio)

CORDIC vs. Other Methods

Feature	CORDIC	Taylor Series	Lookup Tables
Speed	Faster than Taylor for hardware	Slow for real-time	Fast but needs memory
Accuracy	Moderate	High	High
Hardware Feasibility	Excellent	Poor	Moderate

Rotation mode → CORDIC Algorithm

📌 **Objective:** Compute **sine and cosine** using **shift-add operations** (avoiding multiplications).

📌 **Why CORDIC?**

- Efficient for **hardware implementation** (used in FPGA, DSPs).
- Uses only **addition, subtraction, and bit shifts** (good for low-power systems).
- Supports **trigonometric, hyperbolic, and logarithmic functions**.

📌 **Key Idea:**

- Rotate a vector **iteratively** using a **sequence of precomputed angles**.
- If we rotate the unit vector by an angle θ , the final x and y components will give:

$$x_{\text{final}} = \cos(\theta), \quad y_{\text{final}} = \sin(\theta)$$

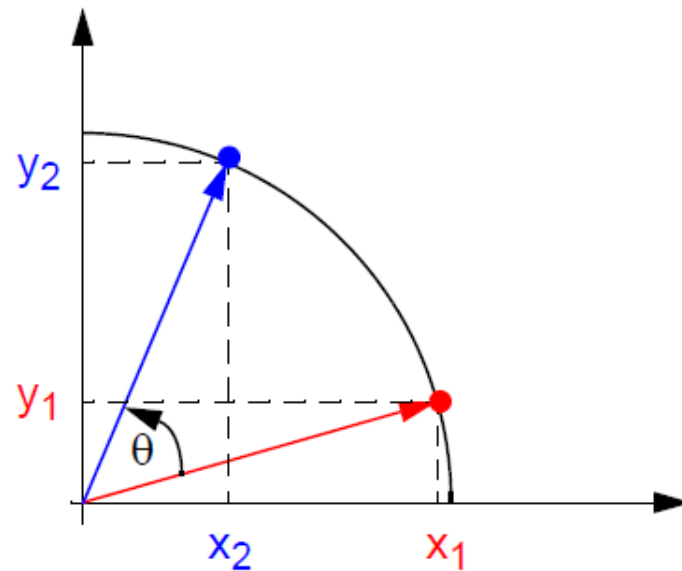
Cartesian Coordinate Plane Rotations

Top
4.2

- The standard method of rotating a point $((x_1, y_1))$ by θ degrees in the xy plane to a point (x_2, y_2) is given by the well known equations:

$$x_2 = x_1 \cos \theta - y_1 \sin \theta$$

$$y_2 = x_1 \sin \theta + y_1 \cos \theta$$



- This is variously known as a plane rotation, a vector rotation, or in linear (matrix) algebra, a Givens Rotation.

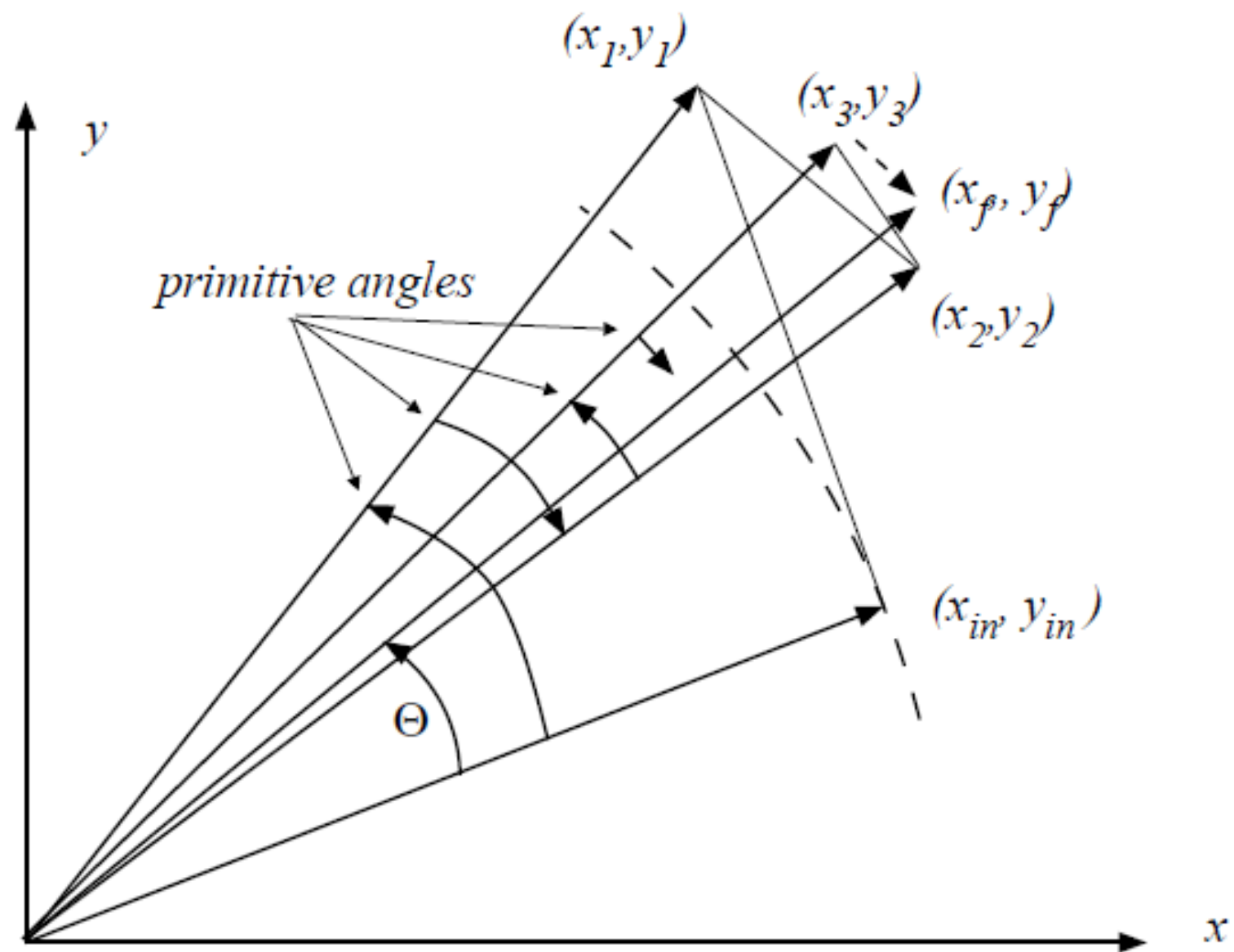


Figure 11.3: Rotating a vector using microrotations.

The table below shows the rotation angles (to 9 decimal places) that can be used for each iteration (i) of the CORDIC algorithm:

i	θ^i (Degrees)	$\tan \theta^i = 2^{-i}$
0	45.0	1
1	26.555051177...	0.5
2	14.036243467...	0.25
3	7.125016348...	0.125
4	3.576334374...	0.0625

i	tanθ	Angle, θ	cosθ
1	1	45.0000000000	0.707106781
2	0.5	26.5650511771	0.894427191
3	0.25	14.0362434679	0.9701425
4	0.125	7.1250163489	0.992277877
5	0.0625	3.5763343750	0.998052578
6	0.03125	1.7899106082	0.999512076
7	0.015625	0.8951737102	0.999877952
8	0.0078125	0.4476141709	0.999969484
9	0.00390625	0.2238105004	0.999992371
10	0.001953125	0.1119056771	0.999998093
11	0.000976563	0.0559528919	0.999999523
12	0.000488281	0.0279764526	0.999999881
13	0.000244141	0.0139882271	0.99999997

$$\cos 45 \times \cos 26.5 \times \cos 14.03 \times \cos 7.125 \dots \times \cos 0.0139 = 0.607252941$$

Angle Accumulator

- The pseudo rotation shown earlier can now be expressed for each iteration as:

$$x^{(i+1)} = x^{(i)} - d_i(2^{-i}y^{(i)})$$

$$y^{(i+1)} = y^{(i)} + d_i(2^{-i}x^{(i)})$$

- At this stage we introduce a 3rd equation called the Angle Accumulator which is used to keep track of the accumulative angle rotated at each iteration:

$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)} \quad (\text{Angle Accumulator})$$

where $d_i = +/- 1$

- The symbol d_i is a decision operator and is used to decide which direction to rotate.

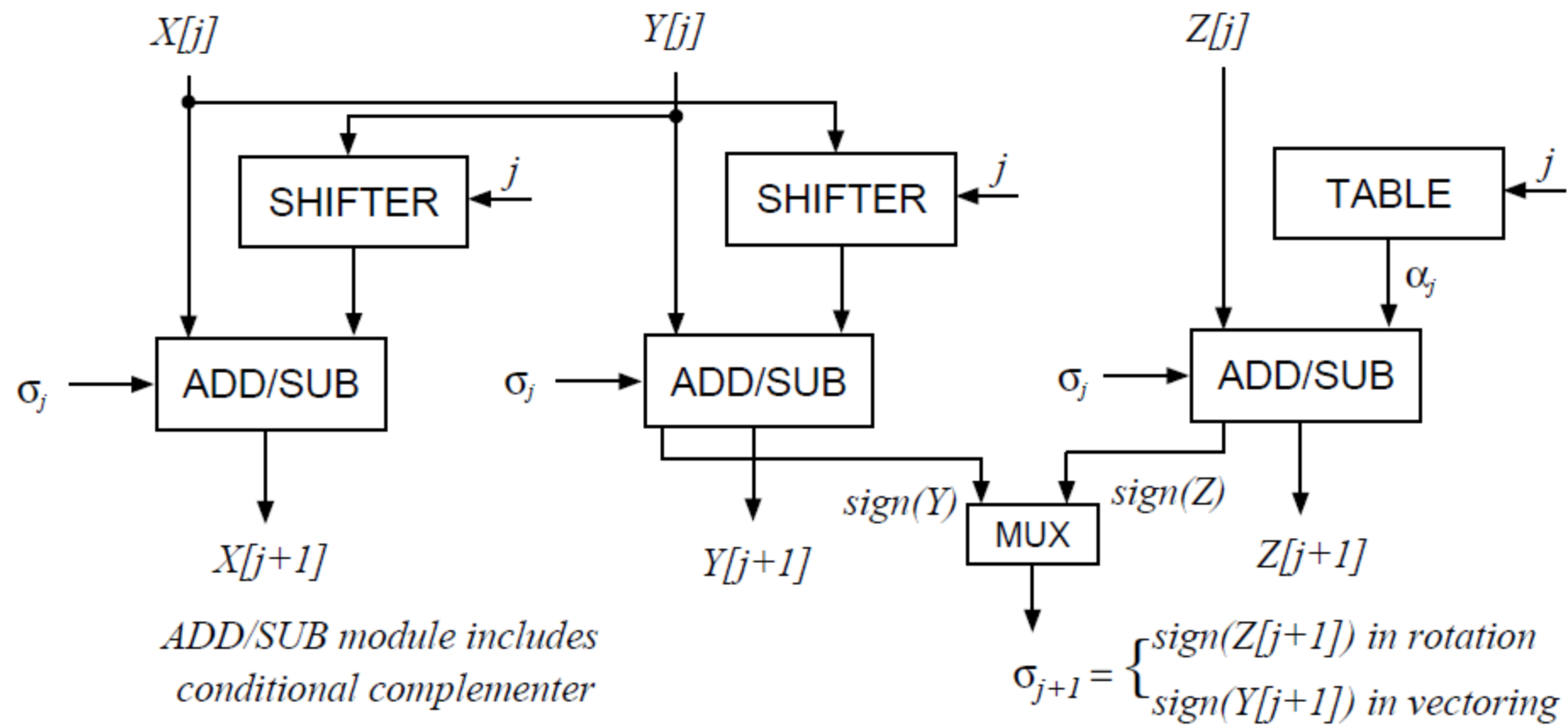


Figure 11.2: IMPLEMENTATION OF ONE ITERATION.

The Scaling Factor

- The Scaling Factor is a by-product of the pseudo-rotations.
- When simplifying the algorithm to allow pseudo-rotations the $\cos\theta$ term was omitted.
- Thus outputs $x^{(n)}$, $y^{(n)}$ are scaled by a factor K_n where:

$$K_n = \prod_n 1/(\cos\theta^{(i)}) = \prod_n (\sqrt{1 + 2^{(-2i)}})$$

- However if the number of iterations are known then the **Scaling Factor** K_n can be precomputed.
- Also, $1/K_n$ can be precomputed and used to calculate the true values of $x^{(n)}$ and $y^{(n)}$.

CORDIC Scaling Factor Formula

The overall scaling factor after n iterations is:

$$K_n = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}}$$

For large n , this converges to a constant:

$$K \approx 0.607252935$$

Effect of Scaling

- The scaling factor K affects the computed values of sine and cosine.
- To obtain correct results, the final computed values must be multiplied by $1/K$.

Compensation for Scaling

- **Precompute** $1/K$ and multiply the output by this value.
- In hardware implementations, this scaling is often ignored for simplicity unless precise results are needed.



Rotation Mode

- The CORDIC method is operated in one of two modes;
- The mode of operation dictates the condition for the control operator d_i ;
- In Rotation Mode choose: $d_i = \text{sign}(z^{(i)}) \Rightarrow z^{(i)} \rightarrow 0$;
- After n iterations we have:

$$x^{(n)} = K_n(x^{(0)} \cos z^{(0)} - y^{(0)} \sin z^{(0)})$$

$$y^{(n)} = K_n(y^{(0)} \cos z^{(0)} + x^{(0)} \sin z^{(0)})$$

$$z^{(n)} = 0$$

- Can compute $\cos z^{(0)}$ and $\sin z^{(0)}$ by starting with $x^{(0)} = 1/K_n$ and $y^{(0)} = 0$

Notes:

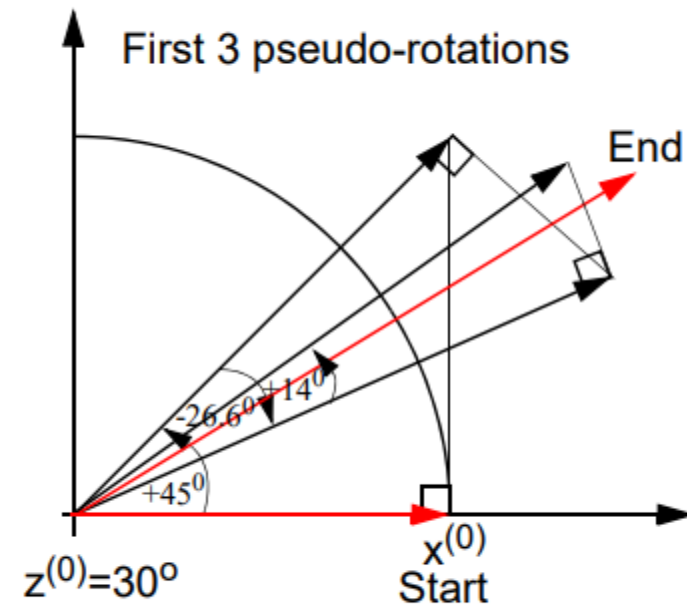
In Rotation Mode the decision operator d_i obeys the condition:

$$d_i = \text{sign}(z^{(i)})$$

Thus, we input $x^{(0)}$ and $z^{(0)}$ ($y^{(0)} = 0$) and then drive $z^{(0)}$ towards 0.

Example: calculate $\sin z^{(0)}$, $\cos z^{(0)}$ where $z^{(0)} = 30^\circ$

i	d_i	$\theta^{(i)}$	$z^{(i)}$	$y^{(i)}$	$x^{(i)}$
0	+1	45	+30	0	0.6073
1	-1	26.6	-15	0.6073	0.6073
2	+1	14	+11.6	0.3036	0.9109
3	-1	7.1	-2.4	0.5313	0.8350
4	+1	3.6	+4.7	0.4270	0.9014
5	+1	1.8	+1.1	0.4833	0.8747
6	-1	0.9	-0.7	0.5106	0.8596
7	+1	0.4	+0.2	0.4972	0.8676
8	-1	0.2	-0.2	0.5040	0.8637
9	+1	0.1	+0	0.5006	0.8657



Vectoring Mode

- In Vectoring Mode choose: $d_i = -\text{sign}(x^{(i)}y^{(i)}) \Rightarrow y^{(i)} \rightarrow 0$
- After n iterations we have:

$$x^{(n)} = K_n \left(\sqrt{(x^{(0)})^2 + (y^{(0)})^2} \right)$$

$$y^{(n)} = 0$$

$$z^{(n)} = z^{(0)} + \tan^{-1} \left(\frac{y^{(0)}}{x^{(0)}} \right)$$

- Can compute $\tan^{-1} y^{(0)}$ by setting $x^{(0)} = 1$ and $z^{(0)} = 0$

Notes:

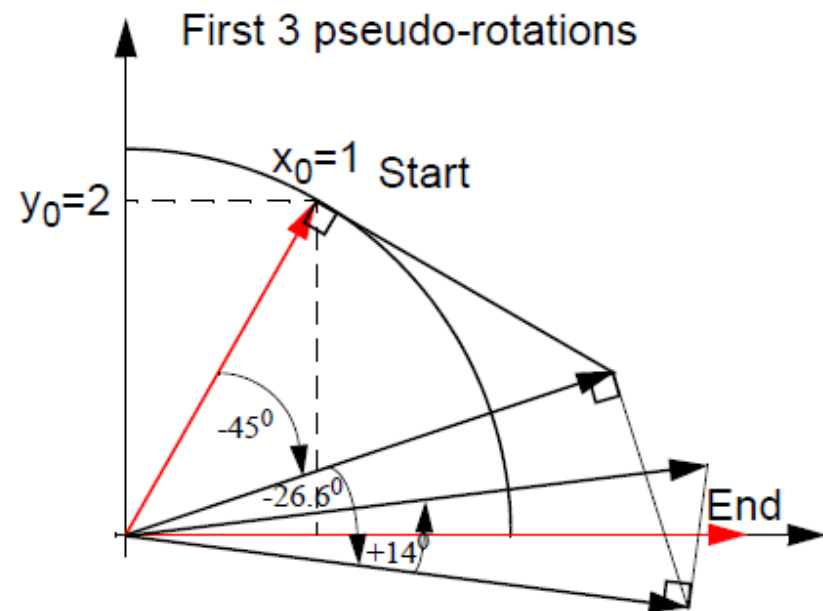
In Vectoring Mode the decision operator d_i obeys the condition:

$$d_i = -\text{sign}(x^{(i)}y^{(i)})$$

Thus, we input $x^{(0)}$ and $y^{(0)}$ ($z^{(0)} = 0$) and then drive $y^{(0)}$ towards 0.

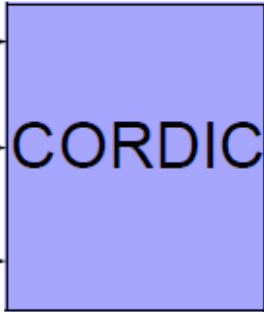
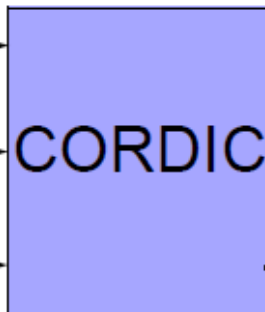
Example: calculate $\tan^{-1}(y^{(0)}/x^{(0)})$ where $y^{(0)} = 2$ and $x^{(0)} = 1$

i	$z^{(i)}$	θ^i	$y^{(i)}$
0	0	45	2
1	45	26.6	1
2	71.6	14	-0.5
3	57.6	7.1	0.375
4	64.7	3.6	-0.078
5	61.1	1.8	0.151
6	62.9	0.9	0.039
7	63.8	0.4	-0.019
8	63.4	0.2	0.009



Circular Coordinate System

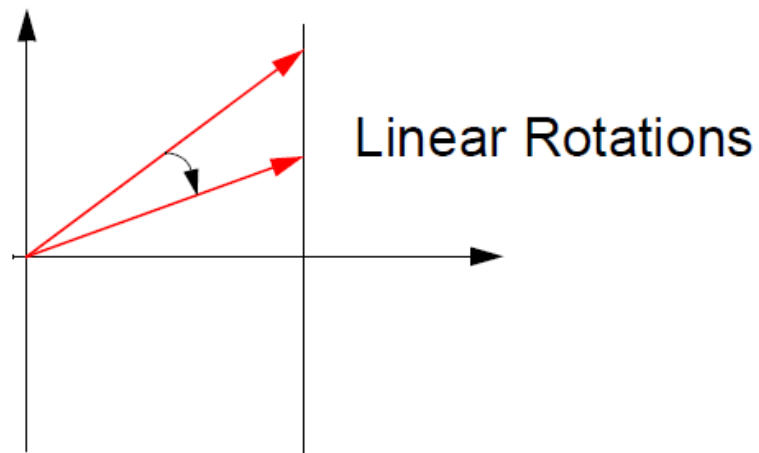
- So far only pseudo-rotations in a Circular Coordinate System have been considered.
- Thus, the following functions can be computed:

Coordinate System	Rotation Mode $z^{(i)} \rightarrow 0; d_i = \text{sign}(z^{(i)})$	Vectoring Mode $y^{(i)} \rightarrow 0; d_i = -\text{sign}(x^{(i)}y^{(i)})$
Circular	<div> $x \rightarrow$ $y \rightarrow$ $z \rightarrow$ </div> <div>  <p>A blue square block labeled "CORDIC" with three input arrows on the left and three output arrows on the right.</p> </div> <div> $K(x.\cos z - y.\sin z)$ $K(y.\cos z + x.\sin z)$ 0 </div> <div> <p>For $\cos z$ & $\sin z$, set $x = 1/K, y = 0$</p> </div>	<div> $x \rightarrow$ $y \rightarrow$ $z \rightarrow$ </div> <div>  <p>A blue square block labeled "CORDIC" with three input arrows on the left and three output arrows on the right.</p> </div> <div> $K(x^2+y^2)^{1/2}$ 0 $z + \tan^{-1}(y/x)$ </div> <div> <p>For $\tan^{-1} y$, set $x = 1, z = 0$</p> </div>

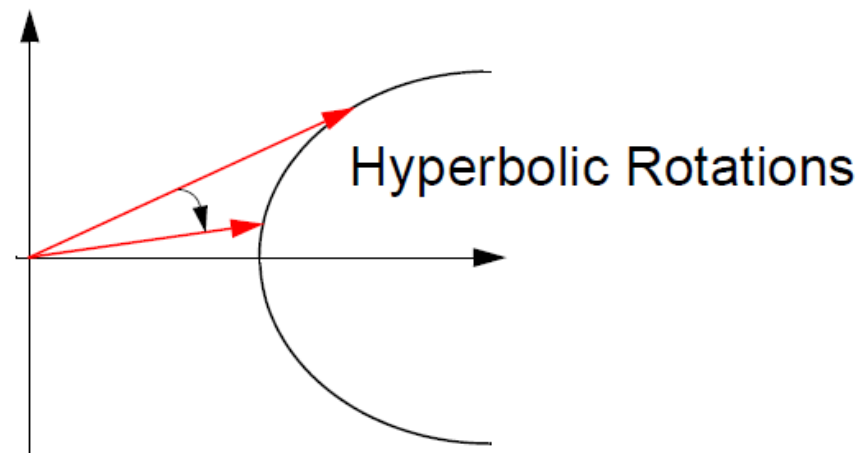
- However, more functions can be computed if we use other coordinate systems.

Other Coordinate Systems

- Linear Coordinate System



- Hyperbolic Coordinate System



Generalised CORDIC Equations

- With the addition of two other Coordinate Systems the CORDIC equations can now be generalised to accommodate all three systems:

$$x^{(i+1)} = (x^{(i)} - \mu d_i (2^{-i} y^{(i)}))$$

$$y^{(i+1)} = (y^{(i)} + d_i (2^{-i} x^{(i)}))$$

$$z^{(i+1)} = z^{(i)} - d_i e^{(i)}$$

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

Summary

CORDIC (COordinate Rotation DIgital Computer) is an iterative algorithm used to compute trigonometric, hyperbolic, and logarithmic functions efficiently using only shifts and additions. It operates in two main modes:

1. Rotation Mode

- Used to compute **sine and cosine** of a given angle.
- Given an initial vector (X_0, Y_0) , it **rotates** it by an angle θ to obtain a new vector.
- Applications: **Trigonometric function evaluation, digital signal processing, and coordinate transformations.**
- Update Equations:

$$X_{i+1} = X_i - d_i Y_i 2^{-i}$$

$$Y_{i+1} = Y_i + d_i X_i 2^{-i}$$

$$Z_{i+1} = Z_i - d_i \theta_i$$

where $d_i = \text{sign}(Z_i)$.

2. Vectoring Mode

- Used to compute the **magnitude** and **angle** of a given vector.
- The goal is to rotate the vector toward the **X-axis** (i.e., make $Y = 0$), computing $R = \sqrt{X^2 + Y^2}$ and $\theta = \tan^{-1}(Y/X)$.
- Applications: **Magnitude computation, complex division, polar-to-cartesian conversion, and signal demodulation.**
- Update Equations:

$$X_{i+1} = X_i + d_i Y_i 2^{-i}$$

$$Y_{i+1} = Y_i - d_i X_i 2^{-i}$$

$$Z_{i+1} = Z_i - d_i \theta_i$$

where $d_i = \text{sign}(Y_i)$.

Both modes avoid multiplications, making CORDIC highly efficient for FPGA and DSP implementations.

i	$\text{atan}(2^{-i})$ (radians)	$\text{atan}(2^{-i})$ in degrees	$\text{atan}(2^{-i})$ in Q1.15 ($\approx \text{atan}(2^{-i}) \times 2^{15}$)
0	0.785398	45°	25735
1	0.463648	26.565°	15192
2	0.244978	14.036°	8027
3	0.124355	7.125°	4074
4	0.062419	3.576°	2045
5	0.031240	1.790°	1024
6	0.015623	0.895°	512
7	0.007812	0.448°	256
8	0.003906	0.224°	128
9	0.001953	0.112°	64
10	0.000977	0.056°	32
11	0.000488	0.028°	16
12	0.000244	0.014°	8
13	0.000122	0.007°	4
14	0.000061	0.003°	2
15	0.000031	0.002°	1

Iteration	Angle (Radians)	Angle (Degrees)	Q1.15 Representation ($\text{angle} \times 32768$)
<code>atan(2^0)</code>	0.7854 rad	45.00°	<code>0.7854 × 32768 = 25736</code>
<code>atan(2^-1)</code>	0.4636 rad	26.57°	<code>0.4636 × 32768 = 15192</code>
<code>atan(2^-2)</code>	0.2449 rad	14.04°	<code>0.2449 × 32768 = 8027</code>
<code>atan(2^-3)</code>	0.1244 rad	7.13°	<code>0.1244 × 32768 = 4074</code>
<code>atan(2^-4)</code>	0.0624 rad	3.58°	<code>0.0624 × 32768 = 2045</code>
<code>atan(2^-5)</code>	0.0312 rad	1.79°	<code>0.0312 × 32768 = 1023</code>
<code>atan(2^-6)</code>	0.0156 rad	0.89°	<code>0.0156 × 32768 = 512</code>
<code>atan(2^-7)</code>	0.0078 rad	0.45°	<code>0.0078 × 32768 = 256</code>
<code>atan(2^-8)</code>	0.0039 rad	0.22°	<code>0.0039 × 32768 = 128</code>
<code>atan(2^-9)</code>	0.0020 rad	0.11°	<code>0.0020 × 32768 = 64</code>
...