Signal Processing Systems (521279S)
Part 3 : CORDIC algorithm and DCT

Version 1.1

November 10, 2025

Goals of the study. Evaluation of functions and transforms is common in signal processing. In this exercise, we study a particular implementation technique, the CORDIC algorithm, which is based on the application of arithmetic shift and addition operations. We consider also signal flow structures for implementing the DCT transform, and how the CORDIC algorithm can be embedded to those structures.

Contents

History
V1.0 11.10.2019 .
V1.1 23.11.2022 Fix an error in Sec. 6.2.3.

1 Introduction

Arithmetic operations studied in the first exercise make up the basis for evaluation of functions. The main approaches for evaluation are

- **approximating functions** which are derived from Taylor series and other expansions. Examples:

$$\begin{aligned} e^x &= 1 + x/1! + x^2/2! + x^3/3! + ... \\ \ln x &= 2(z + z^3/3 + z^5/5 + z^7/7 + ...) \text{ where } z = (x-1)/(x+1) \end{aligned}$$

- **convergence computation** of recursive equation systems: basically two recursive formulas, one formula converges to a constant, the other to the result. Example: if $q^{(0)}$ is an approximation of $\sqrt{x}$, it can refined by evaluation of

$$q^{(i+1)} = 0.5(q^{(i)} + x/q^{(i)})$$

  where i is the iteration index; $q^{(i+1)} - q^{(i)}$ converges to zero.

- **lookup tables** and interpolation

- **coordinate rotation**

The last option is called CORDIC which stands for COordinate Rotation DIgital Computer. The CORDIC principle, which is based on application of shift-add arithmetics[1], can be used for computing a wide range of functions efficiently, and it is well suited to FPGA/ASIC solutions.

2 Givens transform

We consider first computation of the Givens transform, which maps coordinates (x, y) to (x', y') according to the equations

$$\begin{aligned} x' &= x \cos\phi - y \sin\phi \\ y' &= y \cos\phi + x \sin\phi \end{aligned} \tag{1}$$

where ϕ is a rotation angle. This coordinate rotation is a common task in various signal modulators.

[1]Recall from **intro1.pdf** (Sec. 5.1.) the arithmetic shift operation.

2.1 Implementation by multiplications

We note that four multiplications and two additions are needed to compute x' and y' if the values of $\cos\phi$ and $\sin\phi$ are available. With some manipulation, this can also be computed with three multiplications and three additions. Namely,

$$\begin{aligned} x' &= (\cos\phi - \sin\phi)y + \cos\phi(x-y) \\ y' &= (\cos\phi + \sin\phi)x - \cos\phi(x-y). \end{aligned}$$

2.2 Implementation by CORDIC

In the case of CORDIC, multiplication is not used. We develop the algorithm step-by-step in the following.

2.2.1 Concatenation of rotations

Derivation of the CORDIC implementation begins from the note that, if $\phi = \phi_a + \phi_b$, we may first map (x,y) to (x'',y'') using the angle ϕ_a, and then map (x'',y'') to (x',y') using the angle ϕ_b. So, it is possible to concatenate mappings for angles ϕ_i, $(i = 0,\dots,N-1)$ in order to evaluate the mapping for $\phi = \sum_{i=0}^{N-1}\phi_i$.

In the following, we will denote with (x_i,y_i) and (x_{i+1},y_{i+1}) the input and output to the rotation by ϕ_i:

$$\begin{aligned} x_{i+1} &= x_i\cos\phi_i - y_i\sin\phi_i \\ y_{i+1} &= y_i\cos\phi_i + x_i\sin\phi_i. \end{aligned} \tag{2}$$

2.2.2 Applying arithmetic shift and addition

Next, let us assume that $-\pi/2 < \phi_i < \pi/2$. Then, (2) can be rewritten as

$$\begin{aligned} x_{i+1} &= \cos\phi_i(x_i - y_i\tan\phi_i) \\ y_{i+1} &= \cos\phi_i(y_i + x_i\tan\phi_i) \end{aligned} \tag{3}$$

which suggests the computational structure shown in Fig. 1. Note that $\cos\phi_i = \cos(-\phi_i)$ and $\tan\phi_i = -\tan(-\phi_i)$. So, the mapping for a negative angle $-\phi_i$ is the same as for ϕ_i except that signs of the terms involving the tangent are changed.

Multiplication by a power of two corresponds to the arithmetic shift operation which is cheap to implement. The main idea of the CORDIC algorithm is that multiplication by $\tan\phi_i$ can be based on shifting when

$$\tan\phi_i = \pm 2^{-i},\ i \in \{0,1,2,\dots\}.$$

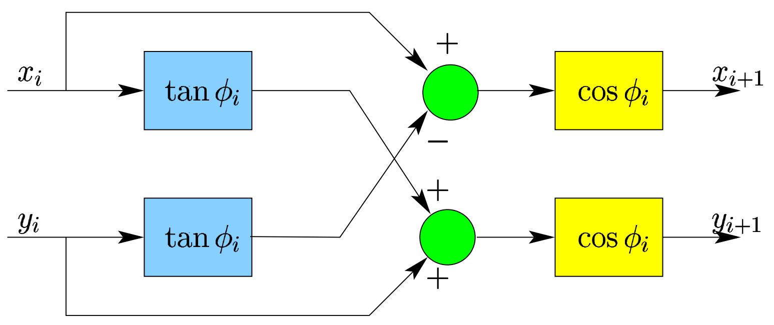


Figure 1: Organizing computations of the Givens rotation transform. For some specific angles ϕ_i, multiplication by $\tan\phi_i$ can be replaced by an arithmetic shift and sign manipulation.

Under this condition, (3) becomes

$$\begin{aligned} x_{i+1} &= \cos\phi_i(x_i - d_i \cdot y_i \cdot 2^{-i}) \\ y_{i+1} &= \cos\phi_i(y_i + d_i \cdot x_i \cdot 2^{-i}) \end{aligned} \tag{4}$$

where $d_i = +1$, if $\phi_i > 0$, and $d_i = -1$, if $\phi_i < 0$. Thus, substituting $d_i = -1$ for $d_i = +1$ corresponds to swapping of signs of the second terms within parentheses, that is, subtraction becomes addition and vice versa.[2]

2.2.3 Gain compensation

However, (4) contains still multiplications by $\cos\phi_i$, and if several rotations were concatenated, we would have lots of multiplications. To solve the problem, we first divide both sides of (4) by $\cos\phi_i = 1/\sqrt{1+2^{-2i}}$, which gives

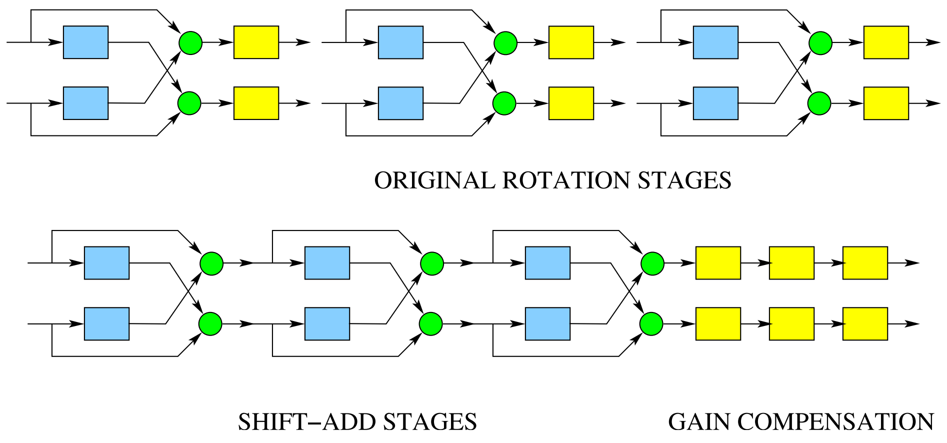
$$\begin{aligned} x_{i+1} \cdot a_i &= x_i - d_i \cdot y_i \cdot 2^{-i} \\ y_{i+1} \cdot a_i &= y_i + d_i \cdot x_i \cdot 2^{-i} \end{aligned} \tag{5}$$

where $a_i = \sqrt{1+2^{-2i}}$. Now, the right hand sides contain just the the shift-and-addition parts of computation, and we get x_{i+1} and y_{i+1} amplified by the gain a_i. We take out the gain blocks from the structure, and get two parts, *shift-add stages* and *gain compensation* as shown in Fig. 2(a). The shift-add stages are controlled by decisions d_i as shown in Fig. 2(b).

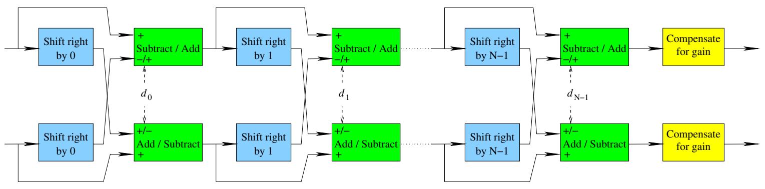
To see, how to compensate for the gain, let us multiply by a constant A_i both sides in (5), and let $A_{i+1} = a_i A_i$. As a result, we get recursive equations

$$\begin{aligned} x_{i+1} \cdot A_{i+1} &= x_i A_i - d_i \cdot y_i A_i \cdot 2^{-i} \\ y_{i+1} \cdot A_{i+1} &= y_i A_i + d_i \cdot x_i A_i \cdot 2^{-i}, \end{aligned} \tag{6}$$

[2]The first rotation angle here is $\phi_0 = 45$ deg. Turkowski (1990) notes that average convergence is slightly better if a one-bit shift to the left is done in the first step ($\phi_0 = \arctan 2^1 \approx 63.4$ deg).



(a) Separating shift-add and gain operations.



(b) Increase in the amount of shift, and control of subtract/add blocks.

Figure 2: Outlines for the CORDIC based rotator.

These equations give the output of a chain of blocks, where just the shift-add parts of the rotations are computed, and multiplications by $\cos\phi_i$ are neglected. After N such steps, we must multiply the results by $1/A_N$ to get x_N and y_N. The value of the gain A_N can be calculated using

$$A_N = \prod_{i=0}^{N-1} \sqrt{1+2^{-2i}}. \tag{7}$$

The value of A_N does not depend on ϕ and is approximately 1.64676 for $N \geq 9$. In practice, compensation can be done somewhere as a pre-processing or post-processing step[3].

2.2.4 Determining rotation directions

The angle ϕ of a composite rotation is uniquely defined by the sequence of elementary rotation directions,

$$(d_0, d_1, \ldots, d_{N-1}).$$

[3]Sometimes CORDIC-like iterative implementation is used also for gain compensation (for example, see (Yu & Swartzlander 2002)).

To determine this sequence on-the-fly at run-time, we need an angle accumulator, that accumulates the elementary rotation angles, and determines in which direction the next rotation should be performed to reduce the angular error. The logic is based on the difference equation

$$z_{i+1} = z_i - d_i \cdot \arctan(2^{-i}) \tag{8}$$

where z_i $(i = 0, 1, \dots)$ denotes the remaining rotation before performing the rotation by ϕ_i $(z_0 = \phi)$. The decision rule is

$$d_i = \begin{cases} -1 & \text{if } z_i < 0 \\ +1 & \text{otherwise} \end{cases} \tag{9}$$

Sequential computation of the shift-add part is illustrated in Fig. 3. The rotation decision block outputs the decisions d_i and remaining angles z_{i+1}.

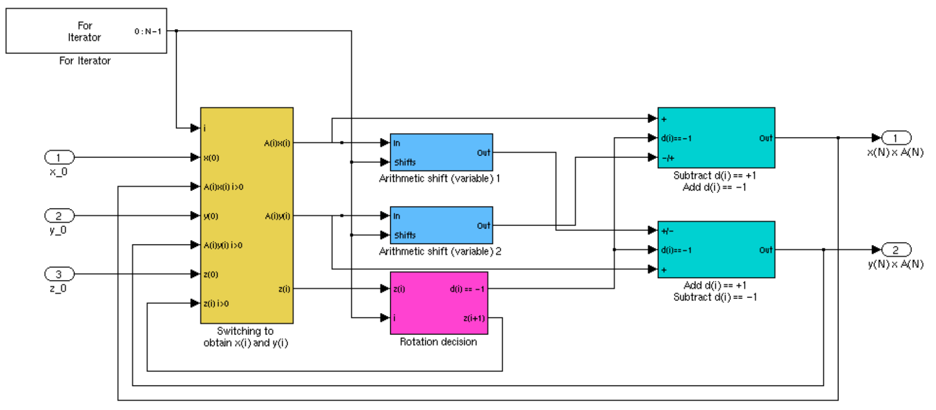


Figure 3: Sequential computation of the Givens transform without gain compensation (from a Simulink model).

Depending on implementation, the angular values $\arctan(2^{-i})$ in (8) may be stored in a small table, or one may use a hardwired solution. Note that if there is only a small number of angles ϕ, extra accumulation/decision component might not be needed as d_i's can be precomputed and tabulated.

Example 1. As a numeric example, consider rotation of (x, y) by the angle of 40 degrees. The decision logic would run as shown in Fig. 4. It can be seen that after seven iterations the remaining rotation would be about -0.49 deg, that is, the result of the corresponding shift/add and gain compensation operations would correspond to the rotation by 40.49 degrees. On average, each rotation improves the accuracy by one bit. However, note that sometimes an iteration can actually increase the error magnitude (e.g. $|z_4|$ is less than $|z_7|$).

Fig. 5 shows the CORDIC rotations for a specific point $(1,0)$. Note how the length of the fraction part increases: for the output of iteration $i = 0$ the format is s2.0, $i = 1$ s3.1, $i = 2$ s5.3, and $i = 3$ already s8.6. For $i = 4$, it would be s12.10 as shift to right by 4 is associated with that iteration. Here, the input has no fraction part: if it contained n bits, then that number of bits should be added to the word and fraction lengths.

Obviously, on a sequential solution (Fig. 3) some fixed word length must be used, and then there is a need to truncate intermediate results. As a result, errors in the final result may be introduced, and it must be confirmed by some means that the accuracy is sufficient.

Finally, Fig. 6 shows the plot of points (x_iA_i, y_iA_i). The initial point $(1,0)$ is on the unit circle, but other points go out of it due to the gains A_i. The point for $i = 4$ is close to the line corresponding to the angle of 40 degrees (as residual z_4 is just 0.40 degrees). ■

i	z_i [deg]	d_i	$\arctan(2^{-i})$ [deg]	z_{i+1} [deg]
0	$+40.00$	$+1$	45.00	-5.00
1	-5.00	-1	26.57	$+21.57$
2	$+21.57$	$+1$	14.04	$+7.53$
3	$+7.53$	$+1$	7.13	$+0.40$
4	$+0.40$	$+1$	3.58	-3.18
5	-3.18	-1	1.79	-1.39
6	-1.39	-1	0.90	-0.49
7	-0.49			

Figure 4: Numeric example of CORDIC iterations.

i	x_iA_i	y_iA_i	d_i	$x_{i+1}A_{i+1}$	$y_{i+1}A_{i+1}$
0	**1**	**0**	$+1$	1 (01.)	1 (01.)
1	1	1	-1	1.5 (01.1)	0.5 (00.1)
2	1.5	0.5	$+1$	1.375 (01.011)	0.875 (00.111)
3	1.375	0.875	$+1$	1.265625 (01.010001)	1.046875 (01.000011)
4	1.265625	1.046875			

Figure 5: First CORDIC iterations according to Fig. 4 for the point $(1,0)$.

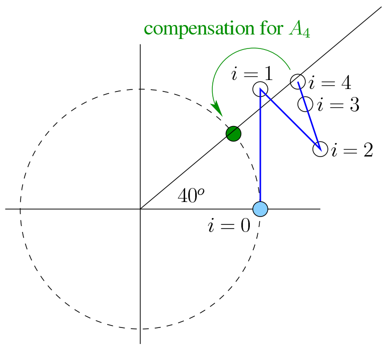


Figure 6: Plotting of points (x_iA_i, y_iA_i) tabulated in Fig. 5. When the coordinates are divided by the gain A_i, we get points (x_i, y_i) on the unit circle.

2.2.5 Extending the range of angles

Note that the method described so far cannot handle the full range of angles

$$\phi \bmod 2\pi \in [0, 2\pi).$$

as the sum of the angles $\arctan(2^{-i})$ cannot reach π (180 deg), because

$$\sum_{i=0}^{\infty} \arctan(2^{-i}) = 45^o + 26.5651^o + ... \approx 99.8830^o.$$

We can see that we cannot reach even 100 degrees.

To extend the range of angles, we can perform an initializing rotation which gives a remaining angle in the range $[-\pi/2, \pi/2]$. For example, let us assume that $\phi \in [-\pi, 0]$. If we add $\pi/2$ to it, we get a value in the range $[-\pi/2, \pi/2]$, which can be used as z_0. But, to compensate for this addition, we must rotate also coordinates (x, y) by $-\pi/2$, and thus the coordinates fed to the CORDIC iterations are now

$$x_0 = x\cos(-\pi/2) - y\sin(-\pi/2) = y$$
$$y_0 = y\cos(-\pi/2) + x\sin(-\pi/2) = -x$$

Likewise, if $\phi \in [0, \pi]$, we subtract $\pi/2$ to get z_0, and compensate for this by rotating coordinates by $\pi/2$.

To combine these ideas, we calculate first the sign of compensating rotation for $\phi \in [-\pi, \pi]$, which is

$$d_{\text{init}} = \begin{cases} -1 & \text{if } \phi < 0 \\ +1 & \text{otherwise.} \end{cases} \tag{10}$$

Initial values for the CORDIC algorithm are then given by

$$\begin{aligned} x_0 &= -d_{\text{init}} \cdot y \\ y_0 &= d_{\text{init}} \cdot x \\ z_0 &= \phi - d_{\text{init}} \cdot \pi/2 \end{aligned} \tag{11}$$

Note that initialization does not introduce any gain ($A_0 = 1$ in recursion). Note also that the method described here is not the only way for performing initialization. See (Andraka 1998) for an alternative.

2.3 Summary

To summarize, the coordinate rotation problem defined in (1) can be solved with the following steps:

1. Represent ϕ in the range $[-\pi, \pi]$.
2. Using (10) and (11), compute the initial values of $x_0 = x_0 A_0$, $y_0 = y_0 A_0$ and z_0.
3. Iterate N times ($i = 0, \ldots, N-1$):
   (a) Determine the rotation direction d_i using (9).
   (b) Compute $x_{i+1}A_{i+1}$ and $y_{i+1}A_{i+1}$ using shift-add arithmetic based on (6).
   (c) Compute the remaining angle z_{i+1} using (8).
4. Compensate for the gain A_N in the result to obtain x_N and y_N.

The iteration loop in the step 3 can be unrolled, which gives a processor consisting of a chain of N units, each performing dedicated iteration. An advantage of such a solution is that shifts are fixed for each unit which allows hardwired implementation of them (Andraka 1998). Unrolled loop can also be mapped to a pipelined implementation.

3 Polar transform

The rotation algorithm described in Sec. 2 can be modified to compute the polar coordinates (r, α) corresponding to particular Cartesian coordinates (x, y). Recall the definition of the Givens transform. The basic idea of polar transform implementation is illustrated in Fig. 7. Using (x, y) as a starting point, rotation ϕ is performed which gives $y' = 0$. When $y' = 0$, ϕ must be $-\alpha$. In addition,

$$\begin{aligned} x' &= x\cos\phi - y\sin\phi \\ &= (r\cos\alpha)\cos(-\alpha) - (r\sin\alpha)\sin(-\alpha) \\ &= r, \end{aligned}$$

so after rotation x' will give the unknown radius. The basic difference with respect to the Givens transform implementation is in the way of making rotation decisions.

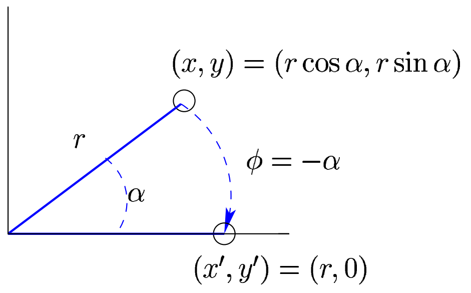


Figure 7: Rectangular to polar transform problem: determine the Givens transform, which maps (x, y) to a point on x-axis.

3.1 Determining rotation directions

Logic for determining the rotation direction d_i must be based on checking the value of y_i. Let us assume that the point (x_i, y_i) lies in the I or IV quadrant, that is, $x_i \geq 0$. We note that if $y_i < 0$ we should rotate counterclockwise, and if $y_i > 0$, we should rotate clockwise. Let z_i correspond to the sum of rotations in clockwise direction, that is, it will converge to α. Then, we have the angle accumulation formula

$$z_{i+1} = z_i - d_i \cdot \arctan(2^{-i}) \tag{12}$$

with the decision rule

$$d_i = \begin{cases} -1 & \text{if } y_i A_i > 0 \\ +1 & \text{otherwise.} \end{cases} \tag{13}$$

3.2 Extending the range of x coordinate

To use the algorithm for negative x, we must perform an initial rotation which gives $x_0 \geq 0$. One approach is just to consider the sign of y. Namely, if $y \geq 0$ and we rotate by $-\pi/2$, then $x_0 = -y \sin(-\pi/2) = y \geq 0$. On the other hand, if $y \leq 0$ and we rotate by $\pi/2$, then $x_0 = -y \geq 0$.

$$d_{\text{init}} = \begin{cases} -1 & \text{if } y > 0 \\ +1 & \text{otherwise.} \end{cases} \tag{14}$$

and the initial values are

$$\begin{aligned} x_0 &= -d_{\text{init}} \cdot y \\ y_0 &= d_{\text{init}} \cdot x \\ z_0 &= -d_{\text{init}} \cdot \pi/2 \end{aligned} \tag{15}$$

With this initialization, the result z_N will converge to α in the range $[-\pi, \pi]$.

3.3 The algorithm

To summarize, the polar coordinates are found as follows:

1. Compute the initial values of $x_0 = x_0 A_0$, $y_0 = y_0 A_0$ and z_0 using (14) and (15).
2. Iterate N times ($i = 0, \ldots, N-1$):
   (a) Determine the rotation direction d_i using (13).
   (b) Compute $x_{i+1}A_{i+1}$ and $y_{i+1}A_{i+1}$ using shift-add arithmetic based on
   $$\begin{aligned} x_{i+1} \cdot A_{i+1} &= x_i A_i - d_i \cdot y_i A_i \cdot 2^{-i} \\ y_{i+1} \cdot A_{i+1} &= y_i A_i + d_i \cdot x_i A_i \cdot 2^{-i}, \end{aligned} \tag{16}$$
   as described in Sec. 2.
   (c) Update the sum of angles, z_{i+1}, using (12).
3. Compensate for the gain A_N in the result to obtain x_N, which corresponds to the unknown radius r. z_N gives the unknown angle α.

Example 2. Let $(x, y) = (3, 4)$. As $y > 0$, $d_{\text{init}} = -1$. First iterations of the algorithm are shown in Fig. 8. The gain is equal to $A_5 = 1.6457$, so $x_5 A_5 / A_5$ gives quite accurately the radius $r = \sqrt{x^2 + y^2} = 5$. Also z_i has converged close to the true α which is $\arctan(y/x) \approx 53.13$ deg. ■

i	x_iA_i	y_iA_i	z_i [deg]	d_i	$\arctan(2^{-i})$ [deg]
0	$+4.00$	-3.00	$+90.00$	$+1$	45.00
1	$+7.00$	$+1.00$	$+45.00$	-1	26.57
2	$+7.50$	-2.50	$+71.57$	$+1$	14.04
3	$+8.13$	-0.63	$+57.53$	$+1$	7.13
4	$+8.20$	$+0.39$	$+50.40$	-1	3.58
5	$+8.23$	-0.12	$+53.98$		

Figure 8: CORDIC iterations for mapping (3,4) to polar coordinates.

4 Unified CORDIC systems and modes

The CORDIC iterations can be implemented in many ways. The algorithms described so far are for *circular* (trigonometric) systems. Similar equations can also be obtained for *hyperbolic* and *linear* systems, and all these systems can be represented with the iteration equations

$$\begin{aligned} x_{i+1} &= x_i - m \cdot d_i \cdot y_i \cdot 2^{-i} \\ y_{i+1} &= y_i + d_i \cdot x_i \cdot 2^{-i} \\ z_{i+1} &= z_i - d_i \cdot \varepsilon_i \end{aligned} \tag{17}$$

where d_i denotes the direction decision, and m and ε_i depend on the system according Table 1.

System	m	ε_i
circular	1	$\arctan(2^{-i})$
hyperbolic	-1	$\operatorname{arctanh}(2^{-i})$
linear	0	2^{-i}

Table 1: Choices of m and ε_i in (17).

There are two modes of computation in this *unified* system,

- *rotation* mode: the sequence of d_i's is such that $z_i \to 0$, and
- *vectoring* mode: d_i's are such that $y_i \to 0$.

Elemental rotations of the hyperbolic system do not converge. However, if certain iterations are repeated convergence is achieved (Andraka 1998). Table 2 shows which computations can be achieved by various combinations of the systems and modes. Some specific cases are considered in the following.

System	Mode	
	rotation	vectoring
circular	init: (x, y, ϕ) $x_i/A_i \rightarrow x\cos\phi - y\sin\phi$ $y_i/A_i \rightarrow x\sin\phi + y\cos\phi$	init: $(x, y, 0)$ $x_i/A_i \rightarrow \sqrt{x^2+y^2}$ $z_i \rightarrow \tan^{-1} y/x$
hyperbolic	init: (x, y, ϕ) $x_i/A_i \rightarrow x\cosh\phi - y\sinh\phi$ $y_i/A_i \rightarrow x\sinh\phi + y\cosh\phi$	init: $(x, y, 0)$ $x_i/A_i \rightarrow \sqrt{x^2-y^2}$ $z_i \rightarrow \tanh^{-1} y/x$
linear	init: $(x, 0, z)$ $y_i \rightarrow x \cdot z$	init: $(x, y, 0)$ $z_i \rightarrow y/x$

• init triplet gives values of (x_0, y_0, z_0)
• the arrow $\rightarrow$ means 'converges to'
• the gain A_i depends on the system and iterations

Table 2: Systems, modes, and associated computations.

4.1 Circular system

The circular system in rotation mode is the algorithm discussed in Sec. 2, and the vectoring mode of the circular system was presented in Sec. 3. The circular system in rotation mode has two notable applications:

- signal modulation (polar-to-Cartesian mapping): in the Givens transform, let x corresponds to an incoming signal $s(n)$, and set $y = 0$. Then,

$$\begin{aligned} x'(n) &= s(n)\cos\phi \\ y'(n) &= s(n)\sin\phi \end{aligned} \tag{18}$$

  It can be seen that x' and y' correspond to modulations of x by cosine and sine waveforms when ϕ is incremented according to a specific angular frequency ($\phi = n\omega$).

- evaluation of sine and cosine functions: if we set $x = 1/A_N$ (gain compensation factor), and $y = 0$ then we can get values of sine and cosine functions by evaluation of shift-add stages; gain compensation is done already in initialization of the algorithm.

4.2 Hyperbolic system

As

$$\exp(\alpha) = \cosh(\alpha) + \sinh(\alpha)$$

the hyperbolic system in rotation mode gives us means to compute the exponential function with initialization $(x_0, y_0, z_0) = (1, 1, \alpha)$. In addition,

$$\ln(\alpha) = 2\tanh^{-1}\frac{\alpha - 1}{\alpha + 1}$$

and therefore the hyperbolic system in vectoring mode can be used for evaluation of logarithms. The same combination can also be used for computing square roots: as

$$(\alpha + 1/4)^2 - (\alpha - 1/4)^2 = \alpha$$

the initialization is $(x_0, y_0, z_0) = (\alpha + 1/4, \alpha - 1/4, 0)$.

4.3 Linear system

The linear system can be used for deriving implementations for multiplication and division. For example, the linear system in rotation mode simplifies the update equations (17) to

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

and with $(x_0, y_0, z_0) = (x, 0, z)$ we get

$$\begin{aligned} y_n &= x \textstyle\sum_{i=0}^{n-1} d_i 2^{-i} \\ z_n &= z - \textstyle\sum_{i=0}^{n-1} d_i \cdot 2^{-i} \end{aligned}$$

As decisions d_i are such that $z_i \to 0$, the sum of bit shifts converges to z, and y_n converges to $x \cdot z$.

5 Three-valued CORDIC

The algorithms described so far use two possible values $\{-1, +1\}$ for each d_i. Such algorithms are said to use *two-valued* CORDIC. However, some iterations may take us farther off from the goal.

Let us consider the circular system with the rotation mode. If we allow skipping some rotation (setting $d_i = 0$), the absolute value of the remaining angle z_i would decrease monotonically. In *three-valued* CORDIC, d_i can take any value from the set $\{-1, 0, +1\}$.

Example 3. Recall Example 1 in Sec. 2. Decision logics for two-valued and three-valued CORDICs would run as shown in Fig. 9. In this case, the 2-valued CORDIC would require seven shift-add stages, whereas the 3-valued solution would need only four *effective* stages. ■

i	2-valued z_i [deg]	2-valued d_i	3-valued z_i [deg]	3-valued d_i	$\arctan(2^{-i})$ [deg]
0	+40.00	+1	+40.00	+1	45.00
1	−5.00	−1	−5.00	0	26.57
2	+21.57	+1	−5.00	0	14.04
3	+7.53	+1	−5.00	−1	7.13
4	+0.40	+1	+2.13	+1	3.58
5	−3.18	−1	−1.45	−1	1.79
6	−1.39	−1	+0.34	0	0.90
7	−0.49		+0.34		

Figure 9: Comparison of convergence of two-valued and three-valued CORDIC algorithms.

However, the gain A_N is *not constant* in the three-valued CORDIC as $\cos 0 \neq \cos(\pm\phi_i)$ which requires special attention in implementation. If the rotation directions d_i are known beforehand then the corresponding gains A_N can be computed beforehand:

$$A_N = \prod_{i=0}^{N-1} a_i, \tag{19}$$

where

$$a_i = \begin{cases} \sqrt{1 + 2^{-2i}} & \text{if } d_i = \pm 1 \\ 1 & \text{otherwise} \end{cases} \tag{20}$$

For example, implementation of the 8-point DCT (discussed in Sec. 4) requires three rotators for fixed angles, and therefore three-valued CORDIC can be utilized to optimize implementation. For an example, see (Yu & Swartzlander 2002).

Note that when a solution for a fixed angle is determined off-line, one does not have to determine rotations sequentially. Instead, *search* for the best combination can be done over all possible sequences of $(d_0, d_1, \ldots, d_{N-1})$ (up to some limit for N).

Example 4. In Example 3 (Fig. 9), rotations for the CORDIC were determined sequentially, and for $N = 7$, the sequence was (+1,0,0,-1,+1,-1,0), which provided the remaining angle $|z_7| = 0.34$ deg. In the case of three-valued CORDIC, we are free to consider other alternatives as illustrated in Fig. 10. ■

i	z_i [deg]	d_i	$\arctan(2^{-i})$ [deg]
0	$+40.00$	$+1$	45.00
1	-5.00	0	26.57
2	-5.00	-1	14.04
3	$+9.04$	$+1$	7.13
4	$+1.91$	0	3.58
5	$+1.91$	$+1$	1.79
6	$+0.12$		

i	z_i [deg]	d_i	$\arctan(2^{-i})$ [deg]
0	$+40.00$	$+1$	45.00
1	-5.00	0	26.57
2	-5.00	0	14.04
3	-5.00	0	7.13
4	-5.00	-1	3.58
5	-1.42	-1	1.79
6	$+0.37$		

i	z_i [deg]	d_i	$\arctan(2^{-i})$ [deg]
0	$+40.00$	0	45.00
1	$+40.00$	$+1$	26.57
2	$+13.43$	$+1$	14.04
3	-0.61		

Figure 10: Some alternative three-valued CORDIC implementations for rotation by 40 degrees (compare to Fig. 9).

Finally note that it may also be beneficial to *repeat* rotation with some specific angle. For example, we can get close to 35 degree rotation, if we rotate twice by 28.08 degrees and once by 7.13 degrees (35.21 degrees total).

6 Implementing DCT with CORDIC

The discrete cosine transform (DCT) is used in many signal processing applications such as image, audio, and video coding. The CORDIC algorithm can be embedded to its implementation. Basically, the DCT transform can be computed by matrix multiplication. However, this computation contains implicit redundancies which are reduced by finding a more efficient *structure* of multiplications and additions. Particular structures allow application of the circular CORDIC (rotation mode) as shown in the following.

6.1 DCT-II / DCT-III

Discrete cosine transform (DCT) (Ahmed *et al.* 1974) and its relative discrete sine transform (DST) are discrete transforms which map vectors of N signal samples to vectors of N transform coefficients. Such a discrete transform can be represented in a matrix form

$$\mathbf{y} = \mathbf{Q}\mathbf{x} \tag{21}$$

where $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]^T$ contains the signal samples, $\mathbf{y} = [y_0, y_1, \ldots, y_{N-1}]^T$ is the vector of transform coefficients, and

$$\mathbf{Q} = \begin{bmatrix} Q_{0,0} & Q_{0,1} & \cdots & Q_{0,N-1} \\ Q_{1,0} & Q_{1,1} & \cdots & Q_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{N-1,0} & Q_{N-1,1} & \cdots & Q_{N-1,N-1} \end{bmatrix}$$

is the transform matrix, whose linearly independent rows form a basis of N-dimensional signal vector space. This is called the *forward transform*, and corresponding *inverse transform* can be defined as

$$\mathbf{x} = \mathbf{Q}^{-1}\mathbf{y}. \tag{22}$$

In DCT, the elements $Q_{k,n}$ are based on cosine functions, and DST uses sine-based functions.

There is a total of eight types of DCT. Only four of them, DCT-I, DCT-II, DCT-III and DCT-IV, are used in practice. Specifically, the transform coefficients y_k $(k = 0, ..., N-1)$ of DCT-II are computed according to

$$y_k = \sum_{n=0}^{N-1} x_n \alpha_k \cos\left(\frac{\pi}{2N}(2n+1)k\right) \tag{23}$$

where x_n refers to data elements and α_k are scaling factors. In the matrix representation (21), the elements of $\mathbf{Q}$ are

$$Q_{k,n} = \alpha_k \cos\left(\frac{\pi}{N}(n+\frac{1}{2})k\right). \tag{24}$$

Scaling factors are typically chosen so that the transform becomes orthogonal, which means that $\mathbf{Q}^{-1} = \mathbf{Q}^T$:[4]

$$\alpha_k = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } k = 0 \\ \sqrt{\frac{2}{N}} & \text{for } k = 1, \ldots, N-1. \end{cases} \tag{25}$$

With this choice, the inverse DCT transform is $\mathbf{x} = \mathbf{Q}^T\mathbf{y}$. Typical choice in applications such as JPEG is to use this orthogonal DCT-II.

The DCT-II is related to the DCT-III which is defined as

$$y_k = \sum_{n=0}^{N-1} x_n \beta_n \cos\left(\frac{\pi}{2N}(2k+1)n\right) \tag{26}$$

where β_n are scaling factors. This transform is orthogonal if

$$\beta_n = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } n = 0 \\ \sqrt{\frac{2}{N}} & \text{for } n = 1, \ldots, N-1. \end{cases} \tag{27}$$

Taking a look at the construction of the transform matrices $\mathbf{Q}$ you can note that the inverse of the orthogonal N-point DCT-II matrix (i.e. transpose) is equal to the orthogonal N-point DCT-III matrix.

6.2 Computing DCT-II

In this section, we take a look at regular structures which can be used for evaluating DCT-II transforms.

6.2.1 Matrix factorizations

The DCT transform involves matrix multiplication, and fast algorithms for such computations can be based on matrix factorization where a multiplier matrix, in our case $\mathbf{Q}$, is represented as a product of two or more matrices. When those matrices are sparse (i.e. they contain lots of zero terms) computationally efficient solutions can be obtained.

First such method for evaluating the DCT-II was proposed by Chen *et al.* (1977). The method has also a recursive structure, which means that N-point DCT is represented in terms of $N/2$-point DCT. The overall structure for an 8-point DCT requires 16 multiplications and 26 additions.

[4]Orthogonal matrices are a special case of *unitary* matrices, which are complex valued matrices, whose inverses are their complex conjugate transposes. If one says that a matrix is orthogonal *up to a scaling factor* it just means that $\mathbf{Q}^{-1} = c \times \mathbf{Q}^T$ for some scalar c $(\neq 1)$.

Various other factorizations for 8-point DCT-II are represented by Feig & Winograd (1992). One particular factorization takes the form

$$\mathbf{Q} = \mathbf{PKB}_1\mathbf{B}_2\mathbf{B}_3$$

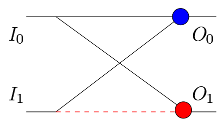
where $\mathbf{P}$ is a signed permutation matrix, $\mathbf{K}$ contains cosine multipliers, and $\mathbf{B}_i$ are all sparse containing only ± 1 non-zero terms (for details, see (Feig & Winograd 1992, Eqs. 16-22)). The method can be implemented with 14 multiplications and 35 additions. One advantage of the approach is that it involves only one multiplication on each computation path. This is significant considering word length requirements for accuracy of computation (Feig & Winograd 1992).

Other methods with reduced number of multiplications have also been derived. However, in many cases those methods have undesirable irregularities in data flow, numerical overflow problems etc. (Loeffler *et al.* 1989). Due to its regularity, Chen's method is still used in software solutions where the number of multiplications is not so critical factor.

6.2.2 Loeffler's structures

Loeffler *et al.* (1989) presented regular computational structures which use only 11 multiplications (theoretical minimum) and 29 additions to compute DCT-II for $N = 8$. Three kinds of elementary structures can be used to build signal flow graphs for those structures:

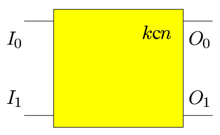
- *butterflies* both add and subtract their inputs:



$$O_0 = I_0 + I_1$$
$$O_1 = I_0 - I_1$$

  Implementation of this operation requires 2 additions;

- *rotators* perform Givens transforms for scaled inputs:



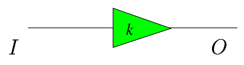
$$O_0 = I_0 k\cos(\pi n/2N) + I_1 k \sin(\pi n/2N)$$
$$O_1 = -I_0 k \sin(\pi n/2N) + I_1 k\cos(\pi n/2N)$$

  Direct implementation of this operation with multipliers requires 3 multiplications and 3 additions, because

$$\begin{aligned} O_0 &= aI_0 + bI_1 = a(I_0 + I_1) + (b-a)I_1 \\ O_1 &= -bI_0 + aI_1 = a(I_0 + I_1) - (a+b)I_0 \end{aligned}$$

where $a = k\cos(\pi n/2N)$ and $b = k\sin(\pi n/2N)$. Note that a, $(b-a)$ and $(a+b)$ are all constants. Note also that there is no scaling if $k = 1$;

- *gain blocks* scale their input,



$O = I\,k$

and require one multiplication.

One structure built from these elements is shown in Fig. 11.[5] All variants presented in (Loeffler *et al.* 1989) require the same number of additions and multiplications. It can be seen that 11 multiplications and 29 additions are required by the structure shown in Fig. 11. To verify this, recall from Sec. 2 that Givens transform can be implemented using 3 multiplications and 3 additions.

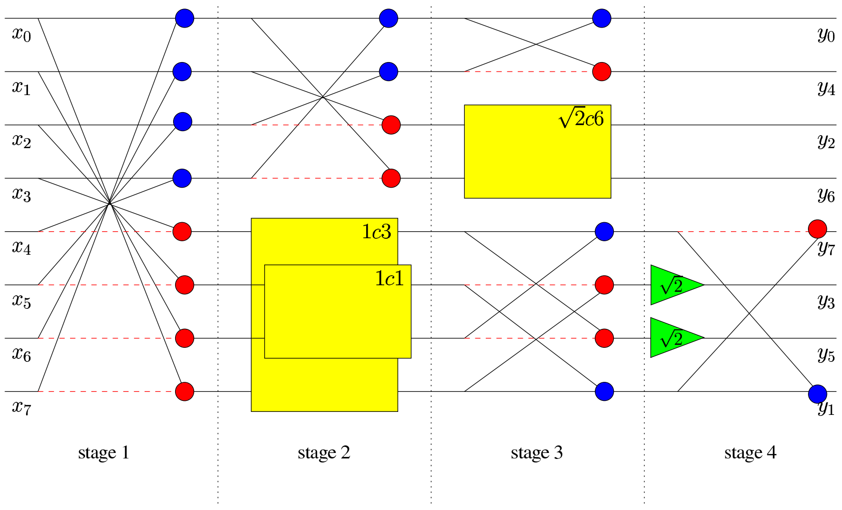
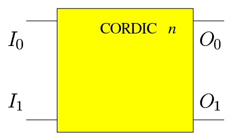


Figure 11: Loeffler's structure for 8-point DCT evaluation.

[5]There seems to be an error in Figure 1 of the Loeffler's paper. The upmost rotation block has the label '$\sqrt{2}c1$' but it should be '$\sqrt{2}c6$'.

6.2.3 Applying CORDIC

As we know, the Givens transform can also be implemented with the CORDIC algorithm. Using just shift-add logic without gain compensation, we get scaled values of the output coordinates. Let us denote the block performing that operation with



$$O_0 = I_0 A\cos(-\pi n/2N) - I_1 A\sin(-\pi n/2N)$$
$$O_1 = I_0 A\sin(-\pi n/2N) + I_1 A\cos(-\pi n/2N)$$

A denotes the gain of the CORDIC shift-add chain

Note that the rotation angle is $-\pi n/16$ and not $\pi n/16$. Using this block, the structure shown in Fig. 11 can be mapped to the structure shown in Fig. 12. Six gain blocks have been added to perform CORDIC gain compensation. We note that complexity of this solution is 18 additions, 8 multiplications + CORDIC implementation overhead. Thus, if this overhead is less than the complexity of implementing 11 multiplications and 29 additions, we have a better design.

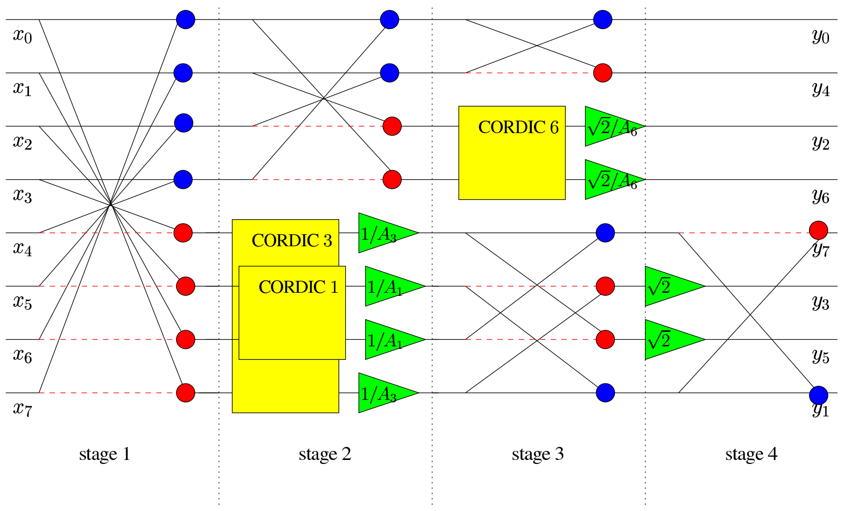


Figure 12: CORDIC embedded to the Loeffler's structure. A_1, A_3 and A_6 denote the gains of CORDIC-1, CORDIC-3 and CORDIC-6 blocks, respectively.

Because the rotation angles are fixed, it is possible to use three-valued CORDIC here (Sec. 3). Recall also that it is possible to implement all gain blocks with shift-add procedures (no multiplication). In some solutions, it may also be possible to allow scaled y_i appear at output and perform gain compensation later[6].

[6]For example, gain compensation can be integrated to the quantization stage in JPEG coding which is one particular application of DCT-II.

6.2.4 Example: JPEG coding

In JPEG coding, an image is divided to 8×8 blocks of pixels. For each 2-D DCT transform is computed, e.g. by applying 1-D DCT to each row first, and then 1-D DCT to each column. So, 8-point DCT must be computed 16 times. According to the standard, orthogonal 8-point DCT-II is to be used.

When 2-D DCT has been computed, the resulting coefficients are quantized (Fig. 13). The quantization is specified by an 8×8 quantization table. It is important to note that the coefficients of a quantization table can be modified to take into account the CORDIC gain. The array produced by 2-D DCT can therefore contain *scaled* coefficients, and the scaling can be non-uniform.

So, assuming that $A_1 = A_3$ in Fig. 12, we can take the gain components out of the structure, which gives the structure illustrated in Fig. 14. We compute the overall gains for each coefficient, and scale the quantization table coefficients accordingly.

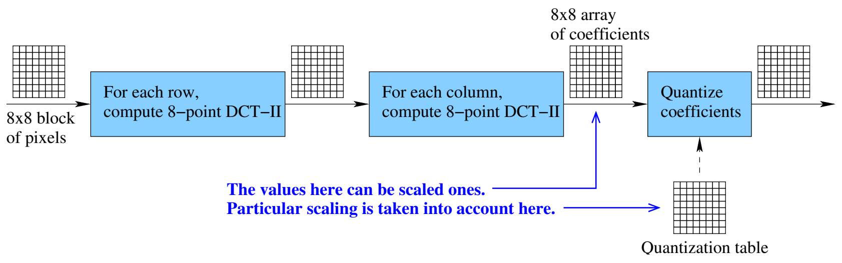


Figure 13: JPEG coding.

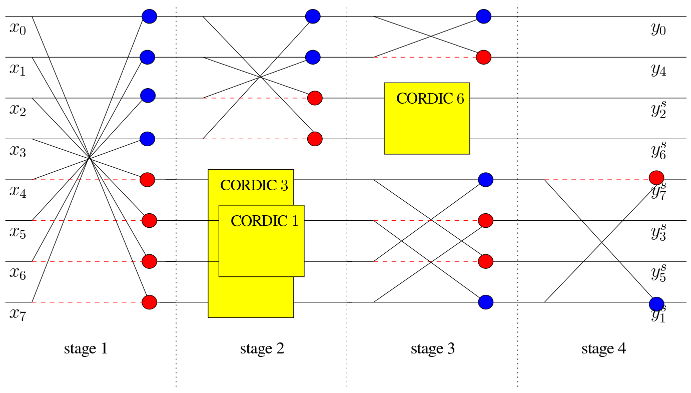


Figure 14: Loeffler's structure with CORDIC blocks, gain corrections removed.

References

Ahmed, N., Natarajan, T. & Rao, K. R. (1974) Discrete cosine transform. IEEE Transactions on Computers C-23(1):90–93.

Andraka, R. (1998) A survey of CORDIC algorithms for FPGAs. In: Proc. ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays. pp. 191–200.

Chen, W. H., Smith, C. H. & Fralick, S. C. (1977) A fast computational algorithm for the discrete cosine transform. IEEE Transactions on Communications COM-25(9):1004–1009.

Feig, E. & Winograd, S. (1992) Fast algorithms for the discrete cosine transform. IEEE Transactions on Signal Processing 40(9):2174–2193.

Loeffler, C., Ligtenberg, A. & Moschytz, G. S. (1989) Practical fast 1-D DCT algorithms with 11 multiplications. In: Proc. International Conference on Acoustics, Speech, and Signal Processing. pp. 988–991.

Turkowski, K. (1990) Fixed-point trigonometry with CORDIC iterations. Apple Computer White Paper.

Yu, S. & Swartzlander, E. E. (2002) A scaled DCT architecture with the CORDIC algorithm. IEEE Transactions on Signal Processing 50:160–167.