Complex CORDIC-like Algorithms for Linearly Constrained MVDR Beamforming

Marius Otte (otte@dt.e-technik.uni-dortmund.de)
Information Processing Lab
University of Dortmund
Otto–Hahn–Str. 4
44221 Dortmund, Germany

Martin Bücker (martin.bucker@research.nokia.com)
Nokia Research Center
Meesmannstr. 103
44807 Bochum, Germany

Jürgen Götze (goetze@dt.e-technik.uni-dortmund.de)
Information Processing Lab
University of Dortmund
Otto–Hahn–Str. 4
44221 Dortmund, Germany

Abstract. In this paper we investigate the use of CORDIC–like approximate rotations for complex valued signal processing applications. In particular we design a processor array for MVDR beamforming with multiple constraints entirely based on linear (Gauss) and circular (Givens) complex transformations. The required transformations are expressed as factorized transformations such that they can be represented as sets of real CORDIC modules. A factorized rotation scheme is applied during the entire algorithm. Each real CORDIC module is replaced by CORDIC–like approximate rotations. The performance of the presented linearly constraint MVDR beamformer is investigated by comparing the bit error rate (assuming 4–QAM modulated signals) and the beampatterns.

Keywords: CORDIC, factorized rotation scheme, approximation, parallel implementation, beamforming

1. Introduction

In recent years signal processing for communications has become one of the main areas of research and development. Especially, the growing number of real time applications in the area of wireless communications requires the development of parallel signal processing algorithms and architectures. Usually, most of the algorithms and architectures presented in the literature assume real input data. This is particularly the case when it comes to the VLSI implementation of the architectures. The complex case is usually covered by standard means (e. g. four real multiplications for the execution of a complex multiplication). In many

practical applications, however, complex data do occur, e. g. , adaptive beamforming [13], multi–user detection [26].

The processor arrays for MVDR adaptive beamforming [19, 18] essentially implement the solution of a complex valued least squares problem, which is obtained by incorporating (one or multiple) constraints in the given minimization problem. Furthermore, a direct computation of the output signal is implemented by a final multiplication of the output signals of the right hand sides by the square root of the conversion factor. The computations for incorporating the constraints are not implemented in these processor arrays but referred to a preprocessing step. The processor arrays are entirely composed of processor cells which evaluate (diagonal cells) or apply (off-diagonal cells) circular rotations.

Therefore, the complexity of the parallel implementations is mainly determined by the complexity of the evaluation and application of the circular (Givens) rotations. Consequently, different strategies, which have been presented for modifying the circular rotations, can be considered in virtue of an efficient implementation:

M1 approximate rotations [21, 2]

M2 factorized rotations [6, 23, 11, 12]

M3 CORDIC [3, 5, 14]

M4 normalized rotations [20]

M5 Combination of the modifications:

- **M1+M2** factorized approximate rotations [8]
- **M1+M3** CORDIC–like approximate rotations [7, 10, 9]

The implementation of two–sided complex rotations based on CORDIC has also been discussed in [16, 15, 25].

In this paper we present an extended processor array for MVDR beamforming with multiple linear constraints. Using the Schur complement the incorporation of the linear constraints into the minimization problem can be formulated as a partial Gaussian elimination. The resulting least squares problem is solved by the QR decomposition, as usual. Even the final multiplication for the direct computation of the output signals can be formulated as a linear transformation by using the Schur complement. Thereby, we achieve an implementation which is entirely based on linear Gauss transformations and circular Givens rotations and can be implemented on an upper triangular array of processors.

In order to implement the complex valued transformations they are formulated in a factorized form. This factorization is composed of a real valued linear and circular transformation, respectively, and phase shifts of the complex numbers involved in the transformation. These phase shifts can also be referred to circular transformations in the complex plane. Therefore, the complex transformations can be referred to a number of real transformations (just like a complex multiplication is composed of four real multiplications). Exploiting these factorizations in detail, however, it is possible to formulate a factorized rotation scheme for a complex transformation, i. e. one of the phase shifts is accumulated in a diagonal matrix which accompanies the involved matrices during the algorithm. This is similar to the ideas of factorized rotations [6, 23, 12] where the scaling factors are swapped out into a diagonal matrix. The diagonal matrix is compensated at the end of the computations. Thereby, the number of required real rotations is reduced and two of the real rotations are just used to annihilate the imaginary parts of the involved complex numbers.

As mentioned above the entire upper triangular processor array for MVDR beamforming with multiple constraints can be implemented based on processor cells executing linear and circular complex 2×2 transformations. These transformations are referred to real linear and circular 2×2 transformations. Each real transformation is implemented using a linear and circular CORDIC processor, respectively. The phase factors accumulated in the diagonal matrix are compensated by the final linear transformations for directly computing the output signal.

We apply CORDIC-like approximate rotations to the real CORDIC modules representing a complex transformation. At first the use of approximate (linear and circular) CORDIC–like rotations for the complex transformations is investigated. Fortunately, all the real transformations, of which a complex transformation is composed of, require the same accuracy of the approximation, i. e. all the real CORDIC–like modules can be implemented using the same number of μ–rotations. Then, CORDIC–like approximate rotations are used for all real CORDIC modules building the (linear and circular) complex transformations in the presented MVDR beamforming processor array.

The presented MVDR beamformer with multiple constraints is applied to a modulated binary signal (assuming 4–QAM). The beampattern and the BER (bit error rate) of the presented MVDR beamforming algorithms and architectures are analyzed in order to obtain a performance profile [22] for the approximation of the CORDIC modules. Different approaches for incorporating the constraints are investigated. Simple methods for finding the shift value (the specific μ–rotation) of the CORDIC–like approximate rotations are investigated. Even the

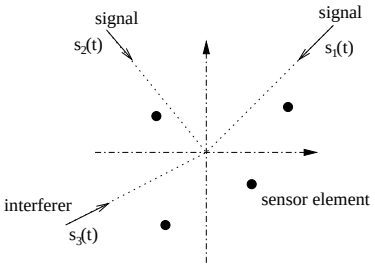


Figure 1. Scenario with four sensor elements, two information bearing signals $s_1(t)$ and $s_2(t)$ from known directions and one interferer $s_3(t)$ from an unknown direction.

use of CORDIC–like modules without scaling factor compensation is discussed. The simulations show, that the required computational effort (approximation accuracy, scaling) strongly depends on the specific conditions of the application (SNR, modulation scheme).

The paper is organized as follows. In section 2 we explain the underlying signal model and review the MVDR beamforming algorithm. An efficient parallel implementation entirely based on complex transformation cells is presented. In section 3 we briefly explain CORDIC and CORDIC-like techniques. The factorized transformations presented in section 4 allow the construction of complex CORDIC-like processor cells in terms of real CORDIC cells. This leads to a very efficient hardware architecture. Results from computer simulations are presented in section 5 to illustrate the performance of the described implementation.

2. MVDR Algorithm

In this section, an adaptive beamformer with minimum variance distortionless response is discussed. Particularly, we focus on its parallel implementation.

2.1. Signal Model

Consider a scenario with M omnidirectional sensor elements located in a plane at positions $\mathbf{m}_i$. The antenna array receives a mixture of desired signals which should be decoded, undesired interferers from unknown directions and background noise equal to all directions. The scenario is depicted in Figure 1. Note, that for the receiving signal $s_1(t)$ the second signal $s_2(t)$ also represents an interferer with known direction of arrival. Whereas $s_3(t)$ is an interferer with unknown direction of arrival. For simplicity, we assume that the direction of propagation is equal at each sensor and the waveforms are planar. Thus, the far-

field assumption holds. The individual sensor outputs $x_i(t)$ should be in baseband form, taking into account the received signals usually are some kind of modulated signals transformed into equivalent lowpass signals. Therefore, we assume a narrow-band approximation, i. e., every antenna element receives the same signal, but delayed in time. The equivalent lowpass signal at sensor i can be written as

$$x_i(t) = \tilde{x}(t) \exp(-j2\pi f_c \tau_i) + n_i(t), \tag{1}$$

where $\tilde{x}(t)$ is the complex baseband signal at a virtual reference sensor element placed in the origin of the antenna coordinate system given in Figure 1, τ_i is the time delay of the signal at sensor i relative to the reference sensor, f_c is the carrier frequency and $n_i(t)$ is white Gaussian noise. Note that equation (1) only holds for exactly one direction. Thus, a complete description of Figure 1 requires two equations. Combining the exp-terms belonging to every antenna and to every known direction of propagation in a $N \times M$ matrix $\mathbf{C}$, with N the number of known signal directions, will allow a more compact notation. The matrix is defined as follows:

$$\mathbf{C} = \begin{bmatrix} e^{j\phi_{1,1}} & e^{j\phi_{2,1}} & \dots & e^{j\phi_{M,1}} \\ e^{j\phi_{1,2}} & e^{j\phi_{2,2}} & \dots & e^{j\phi_{M,2}} \\ \vdots & \vdots & \ddots & \vdots \\ e^{j\phi_{1,N}} & e^{j\phi_{2,N}} & \dots & e^{j\phi_{M,N}} \end{bmatrix}, \tag{2}$$

where $\phi_{i,k} = -2\pi f_c \tau_{i,k}$. The index k corresponds to the kth direction of arrival. For a given array geometry and given directions of arrival, the time delays $\tau_{i,k}$ can be easily calculated by projection considerations.

The discrete signals $x_i(n)$ are considered as output of analog front-ends with one front-end per sensor. Details of the front-end structure are beyond the scope of this paper. The sampled signals are arranged in a $n \times M$ matrix $\mathbf{X}$, with

$$\mathbf{X} = \begin{bmatrix} x_1(1) & x_2(1) & \cdots & x_M(1) \\ x_1(2) & x_2(2) & \cdots & x_M(2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(n) & x_2(n) & \cdots & x_M(n) \end{bmatrix} \tag{3}$$

where n is the number of samples taken at each antenna. Now, weighting the sensor outputs with complex factors w_m, the summation of these products results in a spatial filter, a so-called beamformer [13]. Due to the fact, that every weight vector $\mathbf{w}_i = [w_1, \dots, w_M]^T$ corresponds to one desired output signal $\mathbf{e}_i$, we define a signal matrix $\mathbf{E} = [\mathbf{e}_1 \mathbf{e}_2 \cdots \mathbf{e}_L]$, where $L \leq N$, containing the filter output.

The most straightforward technique for adjusting the weights is to demand that the desired signals should be emphasized, whereas noise and interfering signals propagating from other directions should be suppressed [17]. This leads to the following least squares representation,

$$\min_{\mathbf{w}_i} \|\mathbf{e}_i = \mathbf{X}\mathbf{w}_i^*\|_2^2 \quad \text{for} \quad i \in [1, L] \quad \text{subject to} \quad \mathbf{C}\mathbf{W}^* = \mathbf{B}, \tag{4}$$

where L is the number of desired output signals and $\mathbf{B}$ denotes the gain matrix. Mostly the elements in $N \times K$ matrix $\mathbf{B}$ are taken from the set $\{0, 1\}$; 0 for interference suppression, 1 for unity gain of the information signal. In turn, the weight matrix $\mathbf{W}$ is composed of vectors $\mathbf{w}_i$, such that $\mathbf{W} = [\mathbf{w}_1\mathbf{w}_2 \cdots \mathbf{w}_L]$. So the least-squares criterion tries to minimize the average output of the beamformer and simultaneously fulfilling the constraints. For the scenario depicted in Figure 1 we would obtain $N = 2$ constraints (two known signal directions $s_1(t)$, $s_2(t)$), such that $\mathbf{C} : 2 \times M$, $\mathbf{B} = \mathbf{I}_2$ in order to obtain $\mathbf{E} = [\mathbf{e}_1\mathbf{e}_2]$.

2.2. Solving the Constrained Optimization Problem

One approach to solve the constrained optimization problem is to reformulate the problem as a least squares problem without constraints [13]. With $\mathbf{C} = [\mathbf{C}_1\ \mathbf{C}_2]$, $\mathbf{W}^T = [\mathbf{W}_1^T\ \mathbf{W}_2^T]$ and $\mathbf{X} = [\mathbf{X}_1\ \mathbf{X}_2]$, where $\star = [\star_1\ \star_2]$ denotes the partitioning of matrix $\star$ in two submatices, whereby the first one has N columns, the constraints equation can be written as

$$\mathbf{C}_1\mathbf{W}_1^* + \mathbf{C}_2\mathbf{W}_2^* = \mathbf{B} \tag{5}$$

Solving for matrix $\mathbf{W}_1$, we get

$$\mathbf{W}_1^* = \mathbf{C}_1^{-1}(\mathbf{B} - \mathbf{C}_2\mathbf{W}_2^*). \tag{6}$$

Thus,

$$\begin{aligned} \mathbf{X}\mathbf{W}^* &= \mathbf{X}_1\mathbf{W}_1^* + \mathbf{X}_2\mathbf{W}_2^* \\ &= (\mathbf{X}_2 - \mathbf{X}_1\mathbf{C}_1^{-1}\mathbf{C}_2)\mathbf{W}_2^* - (-\mathbf{X}_1\mathbf{C}_1^{-1}\mathbf{B}) \end{aligned}$$

Therefore, with

$$\overline{\mathbf{X}}_2 = \mathbf{X}_2 - \mathbf{X}_1\mathbf{C}_1^{-1}\mathbf{C}_2 \tag{7}$$

and

$$\overline{\mathbf{B}}_2 = -\mathbf{X}_1\mathbf{C}_1^{-1}\mathbf{B} \tag{8}$$

we have to solve L least squares problems

$$\min_{\mathbf{W}_2} \|\overline{\mathbf{X}}_2\mathbf{W}_2^* - \overline{\mathbf{B}}_2\|_2^2 \tag{9}$$

in order to get $\mathbf{W}_2$ and then calculate $\mathbf{W}_1$ from (6). In a further step we have to calculate the desired output signals by

$$\mathbf{E} = \mathbf{X}\mathbf{W}^*. \tag{10}$$

It is beneficial, particularly with regard to a parallel hardware implementation, to incorporate the steps involved in the solution of the constrained least squares problem (i. e. equations (6), (7), (8), (9), (10)) into one matrix triangularization process.

Let the $(n+N) \times (M+K)$ matrix $\mathbf{M}$ be defined as

$$\mathbf{M} = \left[\begin{array}{c|cc} \mathbf{C}_1 & \mathbf{C}_2 & \mathbf{B} \\ \hline \mathbf{X}_1 & \mathbf{X}_2 & \mathbf{0} \end{array}\right]. \tag{11}$$

Applying a sequence of Gaussian transformations $\mathbf{G}_{pq}(s)$ to the matrix $\mathbf{M}$, where $\mathbf{G}_{pq}(s)$ annihilates the element m_{pq} such that $\mathbf{C}_1$ becomes upper triangular and $\mathbf{X}_1$ is annihilated entirely, results in

$$\mathbf{M}' = \prod_{p,q} \mathbf{G}_{pq}(s) \cdot \mathbf{M} = \left[\begin{array}{c|cc} \mathbf{R}_1 & \overline{\mathbf{C}}_2 & \overline{\mathbf{B}}_1 \\ \hline \mathbf{0} & \overline{\mathbf{X}}_2 & \overline{\mathbf{B}}_2 \end{array}\right], \tag{12}$$

where $\mathbf{R}_1$ is an upper triangular matrix and the $n \times (M-N+K)$ matrix $[\overline{\mathbf{X}}_2\ \overline{\mathbf{B}}_2]$ is the Schur complement of $\mathbf{M}$ [24]. This is actually just a partial Gaussian elimination of $\mathbf{M}$. Note, that we can choose the transformations such that the diagonal elements of $\mathbf{R}_1$ become real.

The least squares problem incorporated in the lower left-hand block can be solved by QR decomposition of $\overline{\mathbf{X}}_2$ in such a way that $\overline{\mathbf{X}}_2 = \left[\begin{array}{cc} \mathbf{Q}_2 & \mathbf{Q}_s \end{array}\right] \left[\begin{array}{c} \mathbf{R}_2 \\ \mathbf{0} \end{array}\right]$, where $[\mathbf{Q}_2\ \mathbf{Q}_s]$ is unitary. Again the required unitary transformation is composed of Givens rotations $\mathbf{G}_{pq}(\theta)$. Defining the matrix $\left[\begin{array}{cc} \mathbf{P}_2^H & \mathbf{P}_s^H \end{array}\right] = \overline{\mathbf{B}}_2^H \left[\begin{array}{cc} \mathbf{Q}_2 & \mathbf{Q}_s \end{array}\right]$ and applying a sequence of Givens rotations to $\mathbf{M}'$ we see that the partial triangularization process of (12) is continued by unitary transformations:

$$\prod_{p,q} \mathbf{G}_{pq}(\theta) \cdot \left[\begin{array}{c|cc} \mathbf{R}_1 & \overline{\mathbf{C}}_2 & \overline{\mathbf{B}}_1 \\ \hline \mathbf{0} & \overline{\mathbf{X}}_2 & \overline{\mathbf{B}}_2 \end{array}\right] = \left[\begin{array}{c|cc} \mathbf{R}_1 & \overline{\mathbf{C}}_2 & \overline{\mathbf{B}}_1 \\ \hline & \mathbf{R}_2 & \mathbf{P}_2 \\ \mathbf{0} & & \\ & \mathbf{0} & \mathbf{P}_S \end{array}\right] \tag{13}$$

As before, the Givens rotations can be chosen such that the diagonal elements of $\mathbf{R}_2$ become real.

By using the equations above, we actually can compute the output signals without computing $\mathbf{W}$:

$$\begin{aligned} \mathbf{E} &= \mathbf{X}\mathbf{W}^* \\ &= \overline{\mathbf{X}}_2\mathbf{W}_2^* - \overline{\mathbf{B}}_2 \\ &= \mathbf{Q}_2\mathbf{R}_2\mathbf{W}_2^* - \overline{\mathbf{B}}_2 \end{aligned} \tag{14}$$

Since $\mathbf{R}_2\mathbf{W}_2^* = \mathbf{P}_2$ and $\overline{\mathbf{B}}_2 = \mathbf{Q}_2\mathbf{P}_2 + \mathbf{Q}_s\mathbf{P}_S$ holds, one obtains

$$\mathbf{E} = -\mathbf{Q}_s\mathbf{P}_S \tag{15}$$

2.3. Processor Array

Since the entire algorithm is formulated as a triangularization of the matrix $\mathbf{M}$, it can be implemented on an upper triangular processor array. In Figure 2 such an array is depicted. The two processor rows at the top (shaded dark) perform the partial Gaussian elimination, while the lighter shaded cells perform the QR-decomposition. In this example we consider a five element sensor array with two signals from known directions impinging on it, hence we have two constraints ($\mathbf{B}$ equals the 2×2 identity matrix). The boundary cells calculate the linear and unitary transformations, respectively. The internal cells carry out the transformations. Figure 3 illustrates the cell functions. The two multipliers at the bottom of the array carry out the multiplication that follow from equation (15). We call it γ-factor compensation. Due to the fact that they destroy the uniform array structure it would be desirable to embed them in the array. Of course, we can use the slightly modificated cell of Figure 3 (b) to do the multiplication. This trivial fact will become more important in our later implementation.

Since the signals are complex valued, the processor cells have to handle complex data. As we see from Figure 3, the cells have to carry out mainly multiplications and summations. However, the diagonal cells have to calculate square roots and/or divisions, respectively. Due to its computational complexity, these computations take more time than the multiplications. To overcome this problem, we will now concentrate on the internal structure of the processor cells.

In virtue of an efficient VLSI implementation, CORDIC processors have been used to implement linear and circular transformations entirely based on shift-and-add operations. In the following we look at the implementation of the complex valued transforms based on CORDIC processors and the use of CORDIC-like approximate rotations.

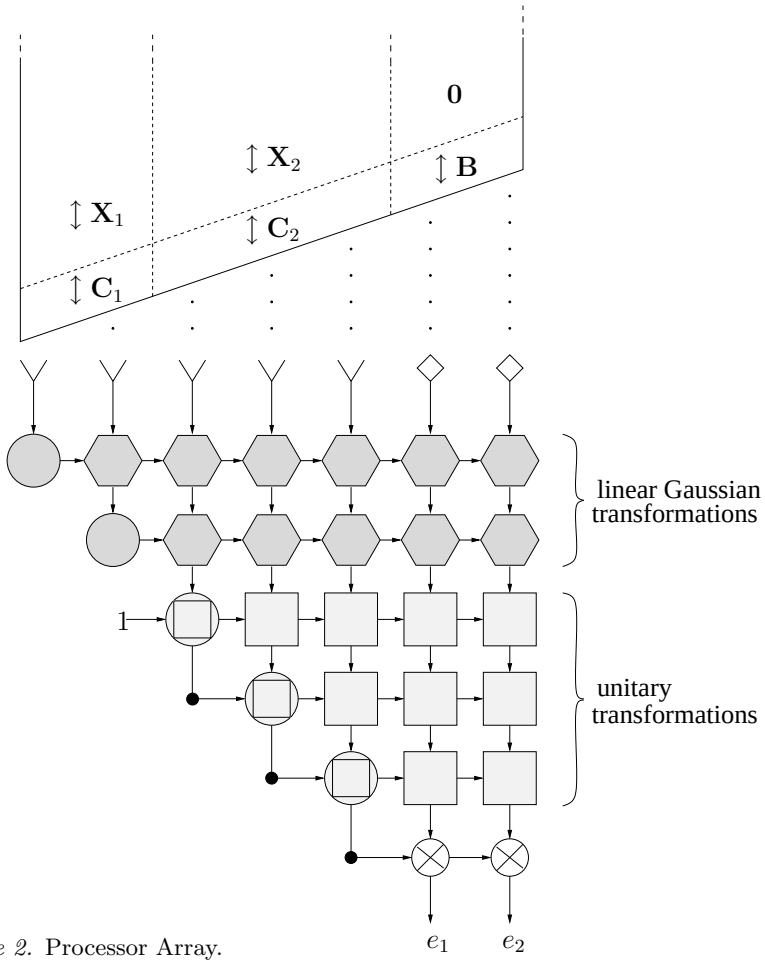


Figure 2. Processor Array.

3. CORDIC-like Algorithms

The CORDIC (*COordinate Rotation on a DIgital Computer*) algorithm makes it possible to carry out vector rotations (and hence to calculate trigonometric functions) only by using shifters and adders, which is very attractive from a hardware point of view. The general CORDIC iteration is given by

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & md_k 2^{-\sigma_k} \\ -d_k 2^{-\sigma_k} & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \tag{16a}$$

$$z_{k+1} = z_k + d_k \alpha_k \tag{16b}$$

Dependent on the choice of m and σ_k we can distinguish between different types of CORDICs. By choosing $m = 1$ and $\sigma_k = k$ we obtain

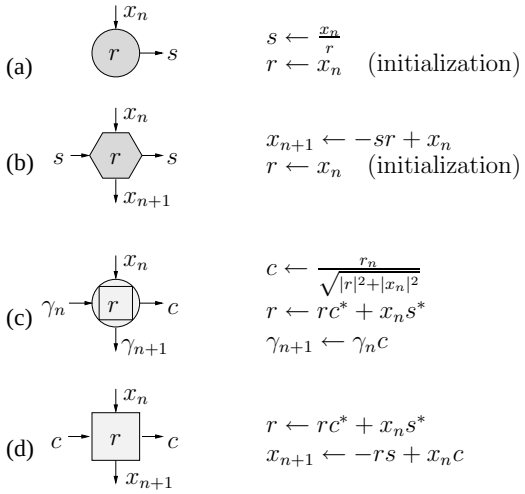


Figure 3. Processor cells.

the classic one originally developed by Volder [27] which can be treated as a scaled rotation of vector $[x_k\, y_k]^T$. That is, if $\mathbf{T}$ corresponds to the exact circular/linear/hyperbolic rotation, $K\mathbf{T}$, $K \in \mathbb{R}$, denotes a scaled rotation. Different values for m and σ_k yield to scaled hyperbolic rotations and linear rotations [28]. The different modes are summarized in Table I.

Table I. CORDIC modes

Mode	m	σ_k	α_k
circular	1	k	$\arctan(2^{-\sigma_k})$
linear	0	k	$2^{-\sigma_k}$
hyperbolic	-1	$1, \ldots, 4, 4, 5, \ldots, 12, 13, 13, 14, \ldots$	$\operatorname{arctanh}(2^{-\sigma_k})$

Moreover the set of possible values for d_k allows us to select either non-redundant ($d_k \in \{-1; 1\}$) or redundant ($d_k \in \{-1; 0; 1\}$) CORDICs.

To perform a complete circular, linear or hyperbolic transformation we have to pass through the iterations in equation (16) until the remaining error fulfills the requirements in some sense. While classical CORDIC computes w micro-rotations (a μ-rotation is one recursion of the full CORDIC sequence, i. e. the execution of equation (16a) with specific m, σ_k, d_k, w is the word length), an approximate CORDIC rotation is defined as one μ-rotation. Note that in approximate signal processing [22] it is often sufficient to break off the iteration after a

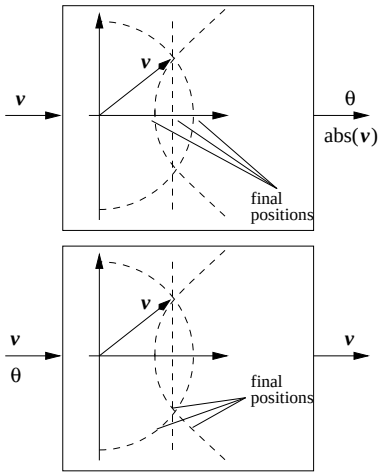


Figure 4. Evaluation (top) and application (bottom) mode of CORDIC. Dependent on the mode (final positions are from left to right: hyperbolic, linear, circular) an exact rotation moves the pointer along one of the dashed lines.

few steps, thus only to apply few μ-rotations. The corresponding exact vector transformation then is given by

$$\mathbf{v}' = K \begin{bmatrix} c(\theta) & s_r(\theta) \\ -s_l(\theta) & c(\theta) \end{bmatrix} \mathbf{v} \tag{17}$$

where $c(\theta)$ denotes $\cos(\theta)$ in circular, $\cosh(\theta)$ in hyperbolic and 1 in linear mode. $s_r(\theta) = s_l(\theta) = \sin(\theta)$ in circular mode, $-s_r(\theta) = s_l(\theta) = \sinh(\theta)$ in hyperbolic mode and $s_r(\theta) = 0$, $s_l(\theta) = \theta$ in linear mode. Furthermore, we can subdivide CORDIC into two operation modes: vectoring (evaluation) and rotation (application) mode. Figure 4 illustrates the modes that have different input and output parameters if treated as black boxes. Now, assume the embedding of CORDIC cells into an array of interconnected cells. Using the non-redundant CORDIC we can easily realize the communication between the cells by transmitting a vector of $[d_1, d_2, \ldots, d_w]$ from one cell to the other. In case of redundant CORDIC, however, we have to transmit the d_k vector as well as the values of $\sigma_k = k$, i, e. the corresponding angles.

The main drawback of CORDIC is that in circular and hyperbolic mode we have to carry out a final multiplication to achieve a scaling factor compensation. Table II lists these scale factors in different modes. For computing the scale factors it is advantageous to use a fixed number of iterations so that the factor can be calculate in advance. Various techniques have been suggested to speed up this scale factor correction [4, 1].

Table II. Scale factors K in different modes.

Mode	K
circular	$\prod_{k=0}^{w-1} \sqrt{1 + d_k^2 2^{-2k}}$
hyperbolic	$\prod_{k=0}^{w-1} \sqrt{1 - d_k^2 2^{-2\sigma_k}}$

Obviously, the maximum rotation angle of CORDIC is limited. The angle cannot exceed $\sum_{k=0}^{w} \alpha_k$. In vectoring mode this is the region of convergence. To overcome this restriction we have to perform a pre-rotation. The pre-rotation has to be as simple as possible. Rotations through 180 degrees (invert $\mathbf{v}$) or through 90 degrees (swap components of $\mathbf{v}$ and invert one) are commonly used. In linear mode, rotations through 180 degrees or simple scalings of y-component are possible.

In the sequel, we use a sequence of CORDIC-like approximate rotations. That is, we have to construct a set of μ-rotations $\{\mu_0, \mu_1, \ldots\}$, where every μ_k corresponds to one tuple $\{d_k, k_k\}$. The crucial point for approximate rotations is to find the approximate angle $d_k \cdot \alpha_k$. Here, we assume a floating point representation of x_k and y_k, respectively. That is, $x_k = m_{x,k} 2^{e_{x,k}}$ and $y_k = m_{y,k} 2^{e_{y,k}}$. The direction of rotation d_k follows from the sign of y_k. The easiest way to obtain $\sigma_k = k$ in linear and circular mode is to subtract the exponents, i. e. to calculate $\sigma_k = e_{x,k} - e_{y,k}$. Other more difficult methods that find the optimal value for σ_k, i. e. the angle $d_k \cdot \alpha_k$ which is closest to the exact rotation angle, can be found in [9]. For methods that apply the scaling of the CORDIC-like approximate rotations the reader is also referred to [9].

4. Complex CORDIC-like Algorithms

In this section we implement complex CORDIC modules in terms of real CORDICs. In order to do so, we first formulate complex rotations in a factorized form. It is shown, that the general complex modules can be simplified in case of using them in the MVDR beamformer.

4.1. Unitary Rotations

Assuming a unitary 2×2 matrix $\mathbf{T}_u$ that is applied to a vector $\mathbf{v} = [r\ a]^T$, where both r and a represent complex numbers; i. e. $r = \Re\{r\} +$

$j\Im\{r\}$ and $a = \Re\{a\} + j\Im\{a\}$. Then $\mathbf{T}_u$ is defined by

$$\mathbf{T}_u = \begin{bmatrix} c^* & s^* \\ -s & c \end{bmatrix}, \tag{18}$$

where $s = a/\sqrt{|r|^2 + |a|^2}$, $c = r/\sqrt{|r|^2 + |a|^2}$ and therefore $\mathbf{T}_u\mathbf{v} = [\sqrt{|r|^2 + |a|^2} \quad 0]^T$. Let φ_a and φ_r the phase angles of a and r, respectively. Now, $\mathbf{T}_u$ can be decomposed into four matrices:

$$\mathbf{T}_u = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & e^{j(\varphi_a+\varphi_r)} \end{bmatrix}}_{\gamma_\varphi} \underbrace{\begin{bmatrix} \cos\delta & \sin\delta \\ -\sin\delta & \cos\delta \end{bmatrix}}_{\mathbf{T}_{3_z}(\delta)} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & e^{-j\varphi_a} \end{bmatrix}}_{\mathbf{T}_2(\varphi_a)} \underbrace{\begin{bmatrix} e^{-j\varphi_r} & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{T}_1(\varphi_r)} \tag{19}$$

where $\sin\delta = \frac{|a|}{\sqrt{|r|^2+|a|^2}}$ and $\cos\delta = \frac{|r|}{\sqrt{|r|^2+|a|^2}}$.

If we now rewrite the two element complex vector $\mathbf{v}$ as a four element vector $\hat{\mathbf{v}}$ with real elements, where

$$\hat{\mathbf{v}} = \begin{bmatrix} \Re\{r\} \\ \Im\{r\} \\ \Re\{a\} \\ \Im\{a\} \end{bmatrix} \tag{20}$$

we can formulate the factorized complex transformation in equation (19) in terms of four real transformations in the following fashion. The application of Givens rotors

$$\tilde{\mathbf{T}}_1(\varphi_r) = \begin{bmatrix} \cos\varphi_r & \sin\varphi_r & 0 & 0 \\ -\sin\varphi_r & \cos\varphi_r & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{21}$$

and

$$\tilde{\mathbf{T}}_2(\varphi_a) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\varphi_a & \sin\varphi_a \\ 0 & 0 & -\sin\varphi_a & \cos\varphi_a \end{bmatrix} \tag{22}$$

on vector $\hat{\mathbf{v}}$ corresponds to the multiplications $re^{-j\varphi_r}$ and $ae^{-j\varphi_a}$, respectively. Likewise, the application of matrix

$$\tilde{\mathbf{T}}_{3_z}(\delta) = \begin{bmatrix} \cos\delta & 0 & \sin\delta & 0 \\ 0 & \cos\delta & 0 & \sin\delta \\ -\sin\delta & 0 & \cos\delta & 0 \\ 0 & -\sin\delta & 0 & \cos\delta \end{bmatrix} \tag{23}$$

on vector $\hat{\mathbf{v}}$ corresponds to the application of $\mathbf{T}_{3_z}(\delta)$ on $\mathbf{v}$. Up to now, we have substituted three of the four complex factors in equation (19). The remaining factor γ_φ can also be expressed as a real rotation. However, it is advantageous not to perform this rotation in combination with the others, but to defer this step. Assuming the phase shifts γ_φ are not performed in combination with the other transformations. That is, there exists a remaining 2×2 diagonal matrix consisting of independent phase factors $e^{j\varphi_1}$ and $e^{j\varphi_1}$, respectively. To perform the next nulling step in the Givens rotations sequence, we have to carry out the unitary transformation given in equation (18). From this it follows that we have to calculate

$$\begin{bmatrix} c^* & s^* \\ -s & c \end{bmatrix} \begin{bmatrix} e^{j\varphi_1} & 0 \\ 0 & e^{j\varphi_2} \end{bmatrix} \begin{bmatrix} r \\ a \end{bmatrix}, \tag{24}$$

where

$$s = \frac{|a|e^{j(\varphi_a+\varphi_2)}}{\sqrt{|r|^2+|a|^2}} \quad \text{and} \quad c = \frac{|r|e^{j(\varphi_r+\varphi_1)}}{\sqrt{|r|^2+|a|^2}} \tag{25}$$

After a few algebraic manipulations we get

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{j(\varphi_a+\varphi_r+\varphi_1+\varphi_2)} \end{bmatrix} \begin{bmatrix} \cos\delta & \sin\delta \\ -\sin\delta & \cos\delta \end{bmatrix} \begin{bmatrix} e^{j\varphi_r} & 0 \\ 0 & e^{j\varphi_a} \end{bmatrix} \begin{bmatrix} r \\ a \end{bmatrix}. \tag{26}$$

The important point lies in noticing that we can accumulate the phase factors in the leading diagonal matrix. And so, the phase compensation can be performed in a final transformation step and need not to be computed in every processor cell. This is similar to the ideas used in factorized rotations [6, 23, 11, 12], where the scalings of the rotations are accumulated in an accompanying diagonal matrix. Since T_u according to (19) can be considered as a factorized rotation, we can handle the first diagonal matrix accordingly. In contrary to factorized rotations, a compensation of the diagonal matrices during the algorithm in order to avoid overflow is not required, since the diagonal matrices γ_φ only contain phase factors.

4.2. Linear Rotations

Similar to unitary rotations, it is possible to write a complex linear rotation as a product of four matrices that are suitable for a real representation. In the linear case, the 2×2 transformation matrix $\mathbf{T}_l$ that is applied to vector $\mathbf{v} = [r\ a]^T$ to annihilate its second component is given by

$$\mathbf{T}_l = \begin{bmatrix} 1 & 0 \\ -\frac{a}{r} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\left|\frac{a}{r}\right| e^{j(\varphi_a-\varphi_r)} & 1 \end{bmatrix}. \tag{27}$$

Like $\mathbf{T}_u$ we define a decomposition of $\mathbf{T}_l$ as

$$\mathbf{T}_l = \underbrace{\begin{bmatrix} e^{j\varphi_r} & 0 \\ 0 & e^{j\varphi_a} \end{bmatrix}}_{\gamma_{\varphi_l}} \underbrace{\begin{bmatrix} 1 & 0 \\ -\left|\frac{a}{r}\right| & 1 \end{bmatrix}}_{\mathbf{T}_{3_l}(|s|)} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & e^{-j\varphi_a} \end{bmatrix}}_{\mathbf{T}_2(\varphi_a)} \underbrace{\begin{bmatrix} e^{-j\varphi_r} & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{T}_1(\varphi_r)}. \tag{28}$$

Comparing the terms in equation (28) with the terms in equation (19) we found that the transformations $\mathbf{T}_1$ and $\mathbf{T}_2$ appear in both cases. The unitary transformation $\mathbf{T}_{3_z}$, however, has to be replaced by a linear transformation $\mathbf{T}_{3_l}$. Again, this complex transformation may be formulated as a 4×4 real transformation matrix:

$$\tilde{\mathbf{T}}_{3_l}(|s|) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -|s| & 0 & 1 & 0 \\ 0 & -|s| & 0 & 1 \end{bmatrix} \tag{29}$$

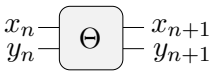
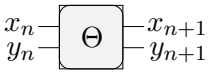
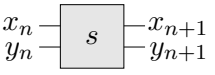
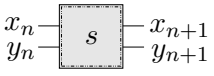
As well as in equation (19) we need to perform a final multiplication with γ_{φ_l}, actually a phase correction of the two elements of $\mathbf{T}_{3_l}\mathbf{T}_2\mathbf{T}_1\mathbf{v}$. Like the foregoing it is possible to inscribe the phase shifts in a diagonal matrix, hence to accumulate them in a separate diagonal matrix.

4.3. Implementation

Now, if we substitute all real transformations for CORDIC blocks the complex CORDIC cells arise from the factorized representations. The complex blocks are displayed in Figure 5. The elementary real CORDIC cells the complex cells based on are listed in Table III.

Recall, that the diagonal values of $\mathbf{R}_1$ and $\mathbf{R}_2$ are real valued, respectively. Hence, we can omit several real CORDIC cells in the above stated complex CORDIC blocks. The simplified blocks used in the array are shown in Figure 6. The complete processor array entirely based on CORDIC cells is shown in Figure 8. Comparing with Figure 2 we have done a few modifications. First, the partial Gaussian transformations to create matrix $[\tilde{\mathbf{C}}\ \tilde{\mathbf{B}}] = [\mathbf{R}_1\ \overline{\mathbf{C}}_2\ \overline{\mathbf{B}}_1]$ out of $[\mathbf{C}_1\ \mathbf{C}_2\ \mathbf{B}]$ are carried out in a second DOA-processing block. That is, due to the high resolution capability of direction of arrival estimations, a better performance can be achieved by calculating the transformations with an increased accuracy. Since the register values in the upper part of the array need not to be updated in every sample step (DOA vary relatively slow with time), the execution time of the DOA estimation is not critical. The values $\tilde{c}_{i,k}$ and $\tilde{b}_{i,k}$ are then assigned to the registers of the linear processor cells. As a consequence, feeding of the constraints matrices into the array is dispensable. This method increases the accuracy of the

Table III. Real CORDIC modules.

Mode	Symbol	Input	Output
circular vector mode		x_n y_n	x_{n+1} $= 0$ y_{n+1} $= 0$ Θ $= \arctan \frac{y_n}{x_n}$
circular rotation mode		x_n y_n Θ	x_{n+1} y_{n+1}
linear vector mode		x_n y_n	x_{n+1} $= 0$ y_{n+1} $= 0$ s $= \frac{y_n}{x_n}$
linear rotation mode		x_n y_n s	x_{n+1} y_{n+1}

constraints in the case of CORDIC-based approximate transformations. Note, however, that since the registers contain $\tilde{c}_{i,k}$ and $\tilde{b}_{i,k}$ the processor array works with a specified accuracy for all CORDIC modules (i. e. fixed number of μ-rotations per CORDIC module). Furthermore, the γ-factor calculation, that was done in the diagonal cells in Figure 2 is now shifted to a new cell column inserted in the array structure. The corresponding cells are shown in Figure 7 (b), (c). Simultaneously, in this column the phase shifts γ_φ and γ_{φ_l} can be carried out.

As mentioned above, we need to replace the final multiplications in Figure 2 with blocks that have a similar internal structure as the rest of the array. Let b_i and c arbitrary complex numbers. By formal application of a Gaussian transformation on $\mathbf{M} = \begin{bmatrix} -1 & [b_1, b_2, \ldots] \\ c & \mathbf{0}^T \end{bmatrix}$ we annihilate c in order to compute the Schur complement of $\mathbf{M}$, i. e.

$$\mathbf{G}_{2,1}(s)\mathbf{M} = \begin{bmatrix} -1 & [b_1, b_2, \ldots] \\ 0 & c[b_1, b_2, \ldots] \end{bmatrix} \tag{30}$$

Obviously, the Schur complement equals the product $c[b_1, b_2, \ldots]$. Therefore, the multiplications may be implemented as a transformation evaluation and application. Fortunately, we can use the same cell for vectoring as we used it in the upper part of the array (subfigure (a) in

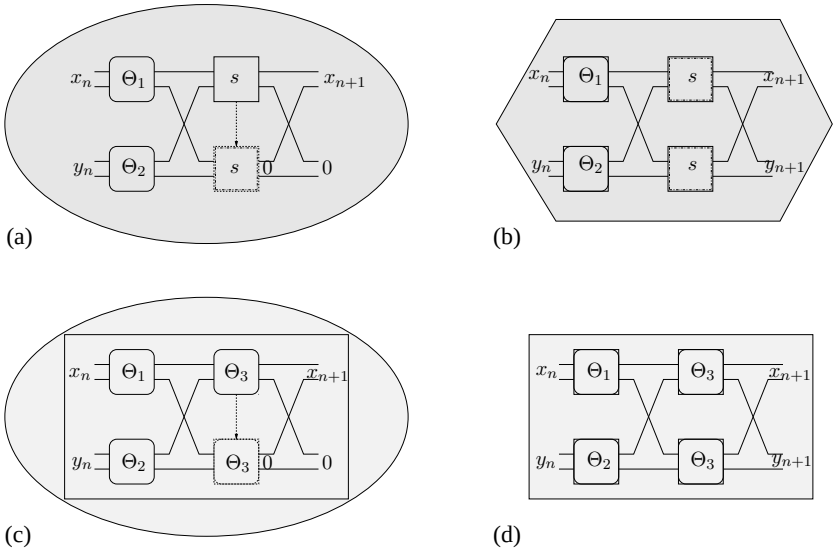


Figure 5. Complex CORDICs for linear evaluations (a), linear application (b), circular evaluation (c) and circular application (d).

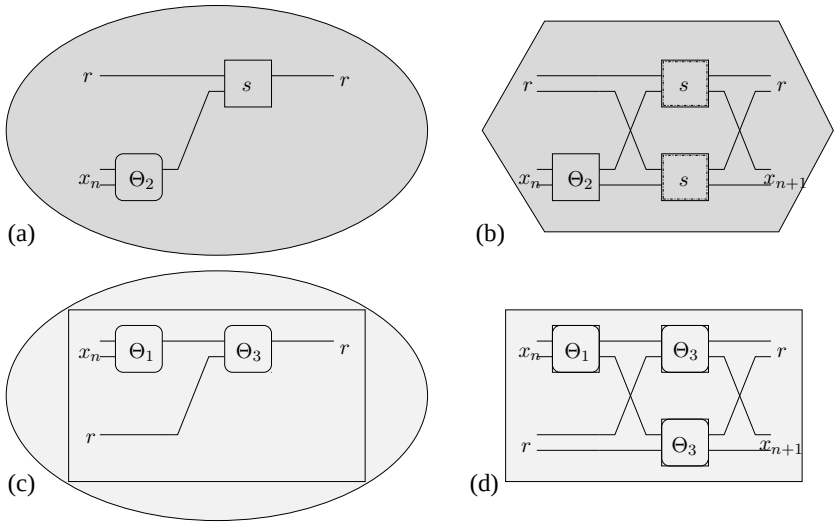


Figure 6. Simplified complex CORDICs for linear evaluations (a), linear application (b), circular evaluation (c) and circular application (d).

Figure 6). Furthermore, the application cell (subfigure (a) in Figure 7) has a very similar structure as the hexagonal cell.

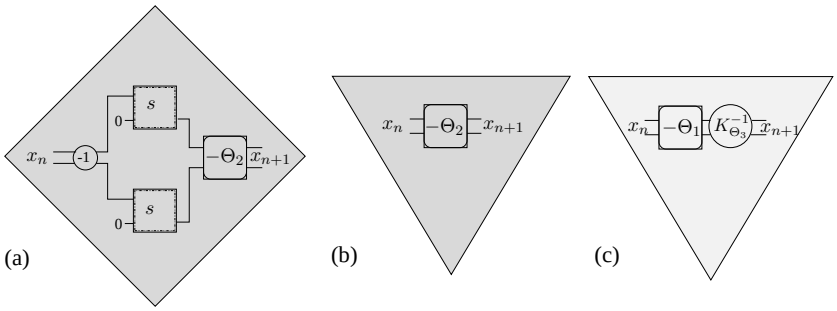


Figure 7. Subfigure (a) shows a CORDIC-based multiplication cell. The triangular cells (b) and (c) perform the γ-factor calculation and accumulate the phase shifts.

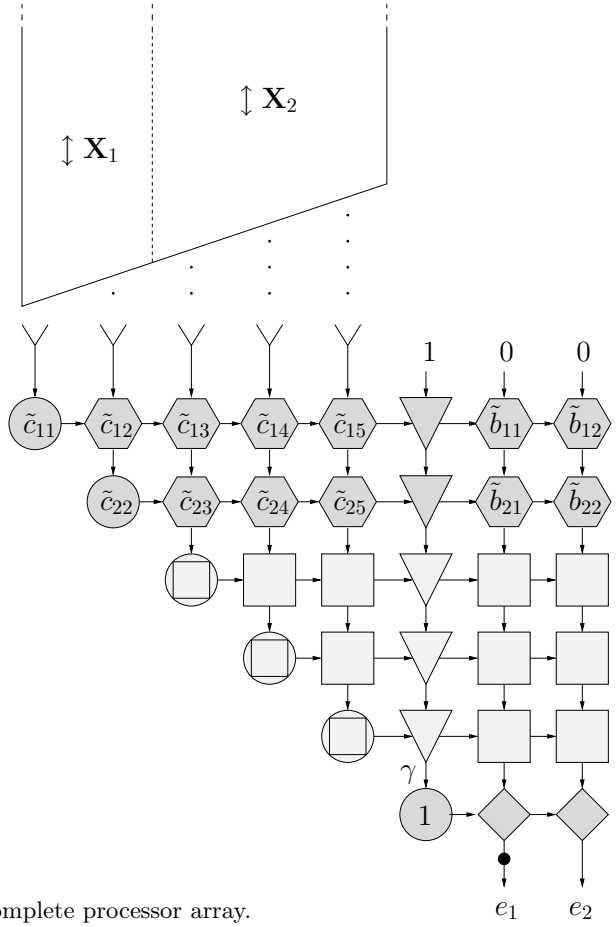


Figure 8. Complete processor array.

5. Simulations

In this section, computer simulations of the proposed array are presented. In the following simulations it is assumed that the signals we want to decode are 4-QAM modulated signals. The parameters have been chosen to represent a scenario with $M = 5$ antennas and 3 impinging signals. The antennas are uniformly distributed in a circle constellation, whereby the distance between adjacent sensors is set to half the wavelength.

The first simulation experiment is conducted with two equi-powered signals impinging from the known directions $-90°$ and $63.4°$, respectively. An interferer with unknown direction is impinging from $128.7°$. The resulting bit error rates as a function of the signal-to-noise ratio are shown in Figure 9. The processor cells work with different number of CORDIC iterations. The required shift values are always calculated by an exponent subtraction (non-optimized angle calculation, see section 3). The results are compared with a simulation trial where exact rotations were used. Notice that already three μ-rotations yield to almost the same bit error rate as the exact computation.

Figure 10 shows BER curves for the same experiment as before, but with scaled rotations. That is, all scale factor compensations are omitted. Even the multiplication cells carry out approximate multiplications. It is important to notice, that even in this coarse approximation the BER converges to the exact curve. In practice, however, we have to decide if the specific problem can cope with scaled rotations (computational performance/ease of implementation versus BER).

We now turn our attention to the resulting beampatterns. To determine the underlying beampatterns of the array we have read out the register cells of the array after certain simulation steps. From this, as discussed in section 2.2, we can calculate the inherent weight vector. The resulting beampatterns are shown in Figure 11. The signal and interferer directions are indicated by dashed lines. The SNR at each antenna is set to 8 dB. Although the curves seem quite different, they meet one's expectation. Both, the exact calculated beampattern as well as the beampatterns of the approximated solutions fulfill the constraint 'amplification equals one' perfectly. Furthermore, the suppression of the known interferer is very accurate. The suppression of the unknown interferer increases with the number of μ-rotations.

To characterize our approach we investigate the quantification of the tradeoff between output quality (BER) and computational effort. For this, Figure 12 shows the performance profile [22] of the array using CORDIC-like approximate rotations with scale factor compen-

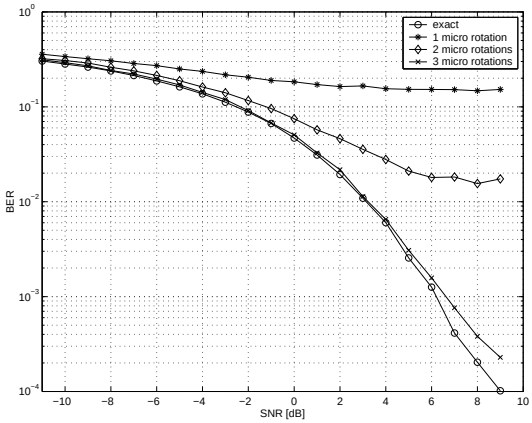


Figure 9. Bit error rate using CORDIC-like approximate rotations with scale factor compensation. Every real CORDIC processor cell executes the same number of CORDIC iterations (μ-rotations). Additionally, BER in case of using exact rotations is given.

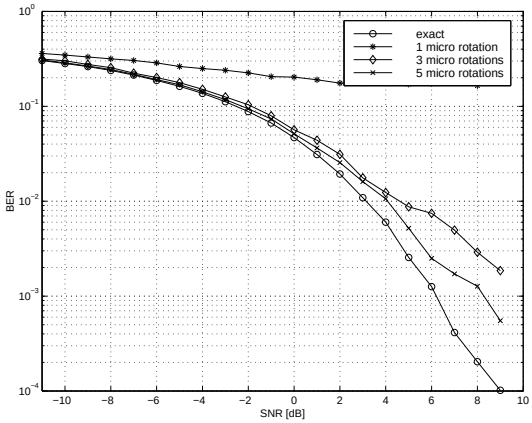


Figure 10. Bit error rate using CORDIC-like approximate rotations without scale factor compensation. Every real CORDIC processor cell executes the same number of CORDIC iterations (μ-rotations). Additionally, BER in case of using exact rotations is given.

sation. Note, that four μ-rotations suffice to achieve the full possible performance.

6. Conclusions

In this paper a systolic processor array for MVDR beamforming with multiple constraints has been presented. The proposed implementation is entirely based on complex linear and unitary rotations. We have de-

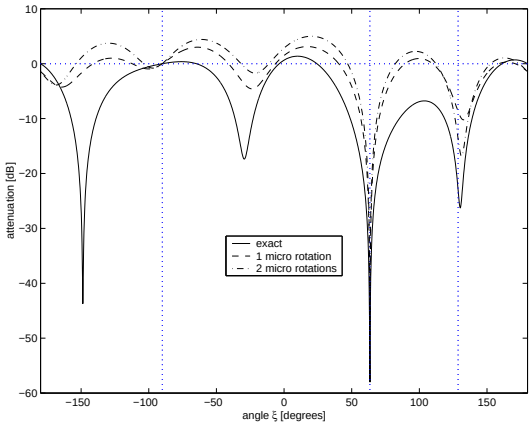


Figure 11. Steered responses extracted from the array (with scale factor compensation). Three signals plus noise are present. Two signals are suppressed, the desired signal power gain is 0 dB. The SNR at each antenna is set to 8 dB. The amplitude function is given by $a(\xi) = 20 \log_{10} |[e^{j\phi_1(\xi)}, \ldots, e^{j\phi_M(\xi)}]\mathbf{w}_1|$.

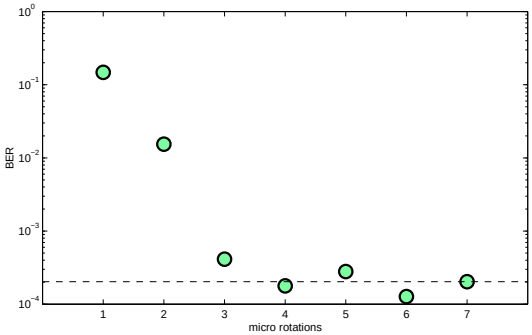


Figure 12. Bit error rate versus number of μ-rotations (SNR = 8 dB). The dashed line indicates BER in case of exact rotations.

signed special CORDIC-based modules for these complex transformations. To reduce the number of operations, some of the involved phase shifts were accumulated into a diagonal matrix and compensated in a final phase compensation multiplication. Furthermore, we employed a non-optimized angle calculation scheme for the real CORDIC-like approximate rotations which is very cheap to compute.