# HOUGH TRANSFORM RECURSIVE EVALUATION USING DISTRIBUTED ARITHMETIC

**Abstract.**

*The Hough Transform (HT) is a useful technique in image segmentation, concretely for geometrical primitive detection. A Convolution-Based Recursive Method (CBRM) is presented for generic function evaluation. In this approach, calculations are carried out by a unique parametric formula which provides all function points by successive iteration. The case of combined trigonometric functions involved in the calculation of the HT is analyzed under this scope. An architecture for reconfigurable FPGA-based hardware, using Distributed Arithmetic (DA) implements the design. It provides memory and hardware resource saving as well as speed improvements according to the experiments carried out with the HT.*

## 1. Introduction

Proposed in 1962, the Hough Transform (HT) has become a widely used technique in image segmentation: plane curve detection [1], object recognition [2], air picture vectorization [3], 3D image reconstruction [4], industrial quality inspection [5], biomedical applications[6][7], quasar reckoning [8], OCR [9], etc. The HT is very suitable because of its robustness, although the great amount of temporary and spatial resources that requires has moved it away from real time applications. This way, the investigation efforts in HT have dealt with the design of fast algorithms and parallel or ad-hoc architectures. As the HT consists of function evaluation using arithmetic operations different algorithmic approaches have been developed: piece-lineal [10], combinatory [11], binary [12], adaptive [13] and fast [14]. There are also implementations of the CORDIC algorithm for applications that demand high speed and precision, such as digital signal and image processing and algebra [15]. However, their drawback is the lower degree of regularity and parallelism capabilities when comparing with the traditional algorithm. The parallelism that underlies in the traditional algorithm allows the implementation of architectures with shared or distributed memory (lineal array, mesh, hypercube [16] and binary tree) as well as specific HT systolic ones [17].

This paper presents the HT calculation under the scope of a Convolution-Based Recursive Method (CBRM), providing a suitable approach for the evaluation of the combined elementary functions involved, sine and cosine, together with the systematic operation of multiplication. Compared with other proposals, CBRM offers good performance in speed, memory requirements and trade-off between precision and error. The *Distributed Arithmetic* implementation provides simplicity to the circuit and also good results as for error and speed.

The paper is structured in seven parts. Following the introduction, the CBRM fundamentals are presented in section 2 and its application to the calculation of the HT is inferred in section 3. Section 4 presents the DA-architecture which is evaluated in section 5. Section 6 compares our proposal with other implementations. Finally, Section 7 summarizes and draws conclusions from this work.

## 2. Convolution-Based Recursive Method (CBRM)

This section introduces some fundamental concepts of the Convolution-Based Recursive Method (CBRM) and outlines its particular features. Mathematical demonstrations that are not under the scope of this paper will be omitted.

Convolution is an operation between two functions that is relevant to many different applications in digital signal and image processing [18], control engineering [19], mathematical morphology [20] or pattern analysis [21]. All these applications share as a common feature the calculation of mathematical spatial or temporary transforms that must be carried out by means of convolution. Generally, convolution is difficult to calculate. This drawback is faced by substituting the initial convolution expression of the functions by the product of these functions which have been previously transformed to frequential ones. However, CBRM is not concerned directly with this well-known duality: it rather exploits the primitive meaning of convolution. That is, the convolution between two functions is a mean to evaluate one of them using the other one as a unit [22]. This fundamental idea provides a powerful basic tool for function evaluation purposes. Equation (1) shows the expansion of convolution in the case of two discrete functions $f$ and $g$ in the interval $[0, +\infty)$. The result of the operation is the function $\Psi$, which represents the evaluation of $f$ by $g$ (or $g$ by $f$).

$$\begin{aligned} &\Psi(0) = f * g(0) = f(0)...g(0) \\ &\Psi(1) = f * g(1) = f(0)\cdot g(1) + f(1)\cdot g(0) \\ &... \\ &\Psi(p) = f * g(p) = f(0)\cdot g(p) + ... + f(p)\cdot g(p) \\ &... \end{aligned} \tag{1}$$

In the case of discretized functions, a recursive formulation of the convolution can easily be achieved

from (1) for function $\Psi$, providing a more compact and useful formula, see equation (2).

$$\Psi_{q+1} = \alpha \cdot \Psi_q + \beta \cdot G_q \qquad q \in [0, I] \subset \mathbb{N} \tag{2}$$

Reciprocally, it can be demonstrated that any discrete recursive equation represented by a weighted sum of two terms like those of equation (2) can be transformed in two convolving discrete functions (namely $f(q) = \alpha^q$ and $g(q) = \beta G_q$, if $\alpha \neq 0$. The equivalence between (1) and (2) provides an approach for discrete function evaluation suitable in many cases (trigonometric, hyperbolic, logarithmic, exponential, inverse, square-root, constant, lineal functions).

The main features of CBRM are:

- CBRM is not concerned with the particular structural characteristics of the evaluated function because it only uses the $\Psi$ value computed at $i$ iteration as an explicit argument which allows the computation of the next value at iteration $i+1$.
- The algorithm runs under a unique compact recursive formula and suits in a great amount of cases.
- The evaluation of any function $\Psi$ is carried out as a sum of two parts which have fixed contributions held by parameters $\alpha$ and $\beta$ (see equation 2). $G$ is an auxiliary function whose value can be evaluated in the same way as $\Psi$ or also can be a function of $\Psi$ itself.
- Parameters $\alpha$, $\beta$ and function $G$ characterize the function $\Psi$ regarding to behavioural aspects and iteration path. Attended that CBRM runs under a unique formula, the combinations of $\alpha$, $\beta$ values are crucial and little changes in any of these values affects dramatically the overall behaviour. Parameter space is tightly related with behavioural features.
- The initial values of $\Psi$ and $G$ are fixed by the user specifications. CBRM provides a new $\Psi$-value per iteration.

A *convolution table* can be built for providing the $(\alpha, \beta, G)$ associated to the calculation of $f * g$, but it must be noticed that in spite of the generic mathematical equivalence between (1) and (2), the computational usefulness of CBRM reduces when function $G$ is not easy to carry out.

## 3. Application of the CBRM to the HT calculation

The geometric primitive detection using the HT implies three stages: image outline creation by using an edge detector, application of the HT to each point of the image and a voting process in the Hough domain in order to extract the geometric primitives.

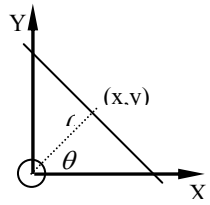


**Fig. 1.** HT parameters for the line detection case

If the geometric primitive to be detected is a straight line, the HT transforms each point $P(x,y)$ in the cartesian domain in a point $(\rho,\theta)$ in the Hough domain, and vice versa, as shown in Fig 1. So, the Hough domain is complete and unique for $0 \leq \rho < \Pi$ line representation.

The Hough domain can be interpreted as a voting grid. Each point in the Cartesian domain votes for a set of lines that intersect it and that stand for a grid point $(\rho,\theta)$. A local maximum point in the voting grid represents the best adjusted line detected. The grid point's increments $\Delta\rho$ y $\Delta\theta$ establish both distance and angular difference between lines in the Cartesian domain, respectively.

The HT is a robust technique since the voting process is not affected by isolated noise points because wrong votes do not affect the local maximum. The HT also manages successfully line occlusion problems, because the distance between points is not relevant.

The parametrized space is discretized in $N_\theta$ levels, from $0$ to $\Pi$ and $N_\rho$ levels, from $\rho_{min}$ to $\rho_{max}$. The HT calculates the $\rho$ values for all the angles in $[0,\Pi[$ and for every pixel in the image. The direct calculation has $O(N^2)$ complexity and the global amount of operations is $N^2 \cdot N_\theta$. If $[0,\Pi[$ is considered as $[0,\Pi/2[ \cup [\Pi/2,\Pi[$, the HT for every pixel $(x_i,y_j)$ in the image can be written as:

$$\begin{aligned} (\rho_I)_i &= x_i \cdot \cos\theta_i + y_j \cdot sen\theta_i \qquad 0 \leq \theta_i < \Pi/2 \\ (\rho_{II})_i &= y_j \cdot \cos\theta_i - x_i \cdot sen\theta_i \qquad \Pi/2 \leq \theta_i < \Pi \end{aligned} \tag{3}$$

If:

$$\begin{aligned} &\theta_k = \theta_{k-1} + \Delta\theta \\ &\cos\theta_k = \cos(\theta_{k-1} + \Delta\theta) \\ &sen\theta_k = \cos(\theta_{k-1} + \Delta\theta) \\ &\cos\Delta\theta = \alpha, \quad sen\,\Delta\theta = \beta \end{aligned} \tag{4}$$

When substituting (4) in (3) we have that:

$$\begin{aligned} &(\rho_I)_i = \alpha \cdot (\rho_I)_{i-1} + \beta \cdot (\rho_{II})_{i-1} \\ &(\rho_{II})_i = \alpha \cdot (\rho_{II})_{i-1} + \beta \cdot (\rho_I)_{i-1} \\ &with \quad \alpha^2 + \beta^2 = 1 \end{aligned} \tag{5}$$

It appears that $(\rho_I)_i$ and $(\rho_{II})_i$ can be crossed-evaluated by applying twice the CBRM equation (2), using $G_q = (\rho_I)_i$ when evaluating $\psi_q = (\rho_{II})_i$ and using $G_q = (\rho_{II})_i$ when evaluating $\psi_q = (\rho_I)_I$, $(\rho_I)_0$ and $(\rho_{II})_0$ should be initialized with the value of the coordinates of each pixel in the image.

## 4. DA-Architecture

The application to the crossed evaluation of $(\rho_I)_i$ and $(\rho_{II})_i$ leads to duplicate the basic architecture of CBRM, as shown in Figure 2, where each $\rho$–value calculation involves two multiplications and one sum.

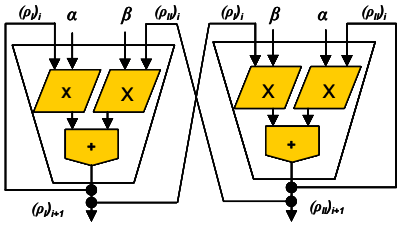


**Fig. 2:** Functional architecture implementing the Hough transform calculation

The proposed implementation of CBRM is based on DA concerned with the trade-off between time delay, memory and hardware resource saving. The main idea consists of taking the partial products of the multiplications performed by the functions $\rho_I$ and $\rho_{II}$ and $\alpha, \beta$, from a *look-up table* (LUT) on every iteration. Due to the fact that $\alpha$ and $\beta$ remain constant through the whole calculation, the following values of $(\rho_I)_i$ and $(\rho_{II})_i$. will access the table in order to provide the partial results. Figure 3 shows the Convolution-LUT table which provides the result pursued in the case of $\rho_I$ It would be a similar one for $\rho_{II}$. A $n$-bit length is assumed for both $(\rho_I)_i$ and $(\rho_{II})_i$ The partial products must be added in order to calculate the final function value at any iteration. These values are feedbacked in order to access the LUTs for a new partial products extraction cycle. We assume a two's complement number coding for partial products in order to allow only add operations. However, before any LUT access, function values must be recoded to the initial signed codification.

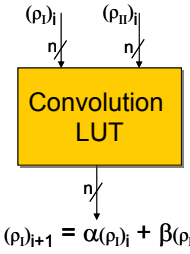


**Fig. 3.** Principle of Convolution-LUT calculation

Memory size involves a limitation for the general case. Therefore, an operator fragmentation into blocks is presented handling a set of bits capable of providing a suitable memory addressing.

Considering that the variables involved in the calculation are $n$-bit length and that $k$ stands for the number of digits of each set, the operators are divided into $t$ parts: $t = n/k$. This way, obtaining the partial results for every block from the memory should be required. On every memory access the operator sign is also pointed out.

Convolution-LUT table memory access for each block is shown in Figure 4.

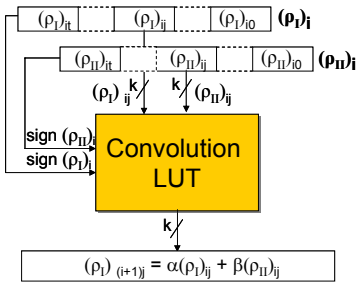


**Fig. 4.** Convolution-LUT partial calculation

It should be pointed out that for $k=1$, the proposed architecture works in a serial way, because the calculation of the $\rho$ functions performs so many LUT accesses as the number of digits of the numbers. In this particular case, the memory structure for function $\rho_I$ is shown in the following table. A similar table can be built for $(\rho_{II})_i$.

| $(\rho_{II})_i\ (\rho_I)_i$ | $\rho_I>0$ $\rho_{II}>0$ | $\rho_I>0$ $\rho_{II}<0$ | $\rho_I<0$ $\rho_{II}>0$ | $\rho_I<0$ $\rho_{II}<0$ |
|---|---|---|---|---|
| **00** | 0 | 0 | 0 | 0 |
| **01** | $\beta$ | $-\beta$ | $\beta$ | $-\beta$ |
| **10** | $\alpha$ | $\alpha$ | $-\alpha$ | $-\alpha$ |
| **11** | $\alpha+\beta$ | $\alpha-\beta$ | $-\alpha+\beta$ | $-\alpha-\beta$ |

**Table 1.** Convolution-LUT structure for $k = 1$

The addition sequence calculation establishes several design possibilities. Data precision is held to $n$:

- *Implementation 1*: Serial scheme for processing each add operands in a single iteration (see Figure 5a).

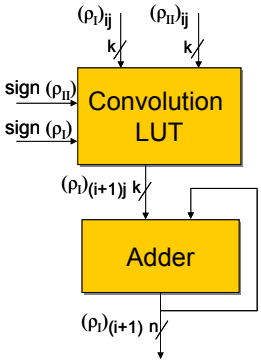


**Fig. 5a.** Serie structure add operands

- *Implementation 2*: Add operand reduction scheme into two final ones so that to perform a final addition. This design is similar to that used in a multiplier for performing partial product reduction (see Figure 5b).

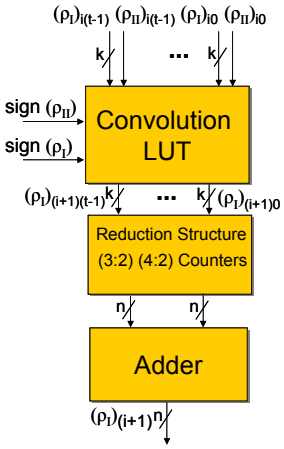


**Fig. 5b.** Reduction structure add operands

This approach provides the whole set of partial add operands in parallel in order to feed a reduction tree process. This way, in spite of a hardware complexity growth, a smaller operation delay is achieved.

*Cycle time*

The delay related to every function value calculation depends on the particular implementation developed. According to the schemes analysed previously, these delays could be expressed in the following way, respectively:

- *Implementation 1:*
  $n$·(Convolution-LUT + Adder)
- *Implementation 2:*
  Convolution-LUT + ReductionStructure + Adder

So, the cycle delay depends on the particular implementation.

## 5. Evaluation of the proposed architecture

In this section, we present area costs and execution time estimates of the architecture proposed in the previous section. The power consumption of the circuit is not important for this research and is therefore not dealt with in this paper. It will be studied in depth in the case that it should be implemented in a chip.

*A. Area estimations*

The model we use for the area estimations is taken from [23], [24] and [25]. The used unit is the size of a complex gate $\tau_a$, since the area of the LUTs and other components are easily expressed in this unit.

- *Convolution-LUT*: Data storage imposes severe restrictions on k block size. As we can see in table 2, the area cost increases exponentially with the k value. Therefore, we have to achieve a balance between the memory required and the complexity of the circuit. The model assumes a 40 $\tau_a$/Kbit rate for tables addressed by words of up to 6-bits long, a 35 $\tau_a$/Kbit rate for 7-11 input bit tables, a 30 $\tau_a$/Kbit rate for 12-13 input bit tables and 25 $\tau_a$/Kbit one for 14-15 input bit tables. Following table shows the cost in terms of $\tau_a$ and bytes for the most common sizes and wordlength.

| n | k | LUT memory requirements | |
|---|---|---|---|
| | | Bytes | Complex gates |
| 16 | 1 | 32 B | 10 $\tau_a$ |
| | 2 | 128 B | 40 $\tau_a$ |
| | 4 | 2 KB | 560 $\tau_a$ |
| | 8 | 512 KB | 102400 $\tau_a$ |
| 32 | 1 | 64 B | 20 $\tau_a$ |
| | 2 | 256 B | 80 $\tau_a$ |
| | 4 | 4 KB | 1120 $\tau_a$ |
| | 8 | 1 MB | 204800 $\tau_a$ |

**Table 2.** LUT memory requirements

- *Adder*: A n-bit adder requires n $\tau_a$ [25]
- *Reduction structure*: A 3:2 counter uses 2 $\tau_a$, and a 4:2 counter uses 4 $\tau_a$.

As shown in the previous tables, the main contributions to the area of the architecture come from the convolution-LUT. With the previous area cost for each component the total area for design alternatives can be calculated.

*B. Delay estimations*

Let $\tau_t$ be the delay of a complex gate, such as one full-adder. The delay estimations have been performed making the following assumptions:

- *Convolution-LUT*: According to [23], [24] analysis[1] we assume a delay of about $T_{LUT} = 3\tau_t$ for 7 input bit tables, $T_{LUT} = 3.5\tau_t$ for 8 input bit tables, $T_{LUT} = 5\tau_t$ for 12-13 input tables and $T_{LUT} = 7.0\tau_t$ for 16-18 input bit tables.
- *Adder*: The delay of the n-bit adder is $\tau_t \log n$
- *Reduction structure:* This module has a delay of 2 $\tau_t$ for 3:2 counter stage and $3\cdot\tau_t$ for 4:2 counter.

## 6. Comparison with other proposals

Two architectures of fast HT based on CORDIC are considered for comparison with CBRM calculation: a pipelined reconfigurable implementation [26] and a parallel one [27]. Area and delay are considered in these comparisons. Error is treated only in the first proposal.

### 6.1. CBRM versus pipelined CORDIC.

In the first proposal [26], a HT using 16-bit fixed-point arithmetic, 12-iteration CORDIC is implemented using a Xilinx XS4010XL-PC84 FPGA for fast prototyping. An FPGA is a reconfigurable logic device consisting of a two dimensional array of RAM-based programmable cells known as *Configurable Logic Blocks* (CLBs). Each CLB consists of a number of function generators implemented as memory LUTs,

[1] Implementation using a family of standard gates from the AMS 0.35 μm CMOS library

storage elements to latch generator results, as well as imputs and outputs. In addition, dedicated carry logic circuitry is available in the CLB's function generators for the fast generation of carry and borrow arithmetic logic to increase the efficiency of adders, subtracters, accumulators, comparators and counters. Programmable routing resources (channels) provide interconnections between the inputs and outputs of configurable elements within an FPGA to appropriate networks. The functionality of each circuit block can be customized via configuration bit-stream.

The Xilinx XS4010XL-PC84 FPGA is a medium capacity device, capable of running at moderate speeds. It has 400 CLBs arranged into 20·20 array, which is equivalent to approximately 10000 gates.

The HT using pipelined CORDIC with serial scale factor compensation uses 83% or 333 CLBs out of 400 of the XS4010XL FPGA. This implementation can be clocked at more than 40 MHz with a computational complexity of $O(N^2)$ for a $N{\cdot}N$ image. At this frequency, a 128·128 binary image with 128 discrete angles ($\Delta\theta$ = 1.40625º) takes 0.0262 seconds to transform one image.

According to the evaluation model presented in section 5 and the characteristics of the Xilinx XC4000 FPGA devices, the following comparison between CBRM and pipelined CORDIC can be drawn, attended that a CLB consists of one LUT-3, two LUT-4 and 2 latches. Data precision is 16 bits and reduction structure chosen is 3:2.

| Pip.CORDIC | Nº CLBs =333 | Complex gates |
|---|---|---|
| **LUT-3** | 1·333=333 | $333{\cdot}2^3{\cdot}2^4{\cdot}40\ \tau_a$ / Kbit $=1665\ \tau_a$ |
| **LUT-4** | 2·333=666 | $2{\cdot}333{\cdot}2^4{\cdot}2^4{\cdot}40\ \tau_a$ / Kbit $=6660\text{-}\tau_a$ |
| **Latches** | 2·333=666 | $2{\cdot}333{\cdot}0.5{\cdot}2^4{\cdot}\tau_a$ $=5328\ \tau_a$ |
| **Overall** | | 13653 $\tau_a$ |

**Tabla 3.** Area estimates for pipelined CORDIC

| CBRM | Implementation 1 | Implementation 2 |
|---|---|---|
| **k=1** | $10\ \tau_a+16{\cdot}1{\cdot}\tau_a=26\ \tau_a$ | $10\ \tau_a+2\ \tau_a+16{\cdot}1\tau_a=28\ \tau_a$ |
| **k=2** | $40\ \tau_a+8{\cdot}2{\cdot}\tau_a=56\ \tau_a$ | $40\ \tau_a+2\ \tau_a+8{\cdot}2\tau_a=58\ \tau_a$ |
| **k=4** | $560\ \tau_a+4{\cdot}4\tau_a=576\ \tau_a$ | $560\ \tau_a+2\ \tau_a+4{\cdot}4\tau_a=578\ \tau_a$ |
| **k=8** | $102400\ \tau_a+2{\cdot}8{\cdot}\tau_a=$ $102416\ \tau_a$ | $102400\ \tau_a+2\ \tau_a+2{\cdot}8\tau_a=$ $102418\ \tau_a$ |

**Table 4.** Area estimates for CBRM

CBRM occupies an area increasing with the block length $k$, but its implementations are better than the pipelined CORDIC ones up to $k$=8, as shown in Tables 3 and 4.

For delay estimation, the HT calculation by CBRM involves 64·128·128 iterations and each iteration has $16/k$ cycles. Assuming that $\tau_t \approx 1$ ns in the XS4010XL-PC84 FPGA.

| CBRM | Implementation 1 | Implementation 2 |
|---|---|---|
| **k=1** | $16{\cdot}64{\cdot}128{\cdot}128{\cdot}(3\tau_t+\tau_t \lg 16)$ =117,440 ms | $16{\cdot}64{\cdot}128{\cdot}128{\cdot}(3\tau_t+2\tau_t)+\tau_t \lg 16$ =83,886 ms |
| **k=2** | $8{\cdot}64{\cdot}128{\cdot}128{\cdot}(3\tau_t+\tau_t \lg 8)$ =50,332ms | $8{\cdot}64{\cdot}128{\cdot}128{\cdot}(3\tau_t+2\tau_t)+\tau_t \lg 8$ =41,943 ms |
| **k=4** | $4{\cdot}64{\cdot}128{\cdot}128{\cdot}(3.5\tau_t+\tau_t \lg 4)$ =23,069 ms | $4{\cdot}64{\cdot}128{\cdot}128{\cdot}(3.5\tau_t+2\tau_t)+\tau_t \lg 4$ =23,068 ms |
| **k=8** | $2{\cdot}64.128{\cdot}128{\cdot}(5\tau_t+\tau_t \lg 2)$ =12,583 ms | $2{\cdot}64{\cdot}128{\cdot}128{\cdot}(5\tau_t+2\tau_t)+\tau_t \lg 2$ =14,680 ms |

**Table 5.** Delay estimates for CBRM

Table 5 shows that delays are better for implementation 2 of CBRM than for implementation1 up to $k$=4. For $k$=4 and $k$=8 both CBRM implementations have better results than pipelined CORDIC (0.0262 s).

In the pipelined CORDIC implementation, the global error ($E = 2N \cdot 2^{-(n-1/2)} + 2^{-M} \cdot n$.) falls with the number of bits of the fractional part M and grows with the number of iterations, $n$, when $n>16$. According to [26], the absolute error for $N=128$ is 0.135, for 16 bits data (with 8 fractional) and 12 iterations. It is the same than 6 erroneous fractional bits in the result. That represents an approximate relative error of $2^{-5}$=3%. In this paper, the complete error results for CBRM are not presented. Table 6 only summarizes absolute error results for some $\Delta\theta$ values versus the number of calculated points. Data precision is 32 bits.

| | Number of calculated points (I) | |
|---|---|---|
| | **I= 12** | **I= 36** |
| **Δθ = π/4 rad** | $3.52\ 10^{-6}$ | $9.41\ 10^{-6}$ |
| **Δθ = π/72 rad** | $1.17{\cdot}10^{-5}$ | $6.17{\cdot}10^{-5}$ |
| **Δθ = π/360 rad** | $5.92{\cdot}10^{-5}$ | $9.51{\cdot}10^{-5}$ |

**Table 6.** Error estimates for CBRM

It can be noticed that CBRM results seem to be better than pipelined CORDIC ones for similar Δθ values ($\Delta\theta$ = 1.40625º lays between π/72 rad and π/360 rad), although it is not an easy comparison because of the precision of data, 16 bits for CORDIC, 32 for CBRM. We can also observe that CBRM error increases with the number of calculated points. So, in this case it would be necessary to implement with a parallel CBRM in order to hold the error.

### 6.2. Parallel CBRM versus parallel CORDIC.

The second proposal [27] considers a parallel implementation of the CORDIC algorithm in order to calculate the HT. The computation of the HT of an $N{\cdot}N$ image with single CORDIC processor requires $N^3/2$ cycles, assuming that in each evaluation two values for parameter $\rho$ are obtained. The computation time can be reduced by means of parallelism. Three possible approaches are considered, namely parallelization of the pixels in the image, of the angle $\theta$, or both simultaneously. The latter requires $N^3/2$ CORDIC processors, one processor per pixel per angle; the evaluation of the transform takes only the time of one CORDIC operation ($n$ cycles for radix 2, $n/2 + n/4$ cycles for mixed radix 2-4 and $n/2$ for radix

4, $n$ is the data precision) but the hardware is considerably increased. Also conflicts occur in votation process because the results obtained by the processors with the same angle $\theta$ can vote over the same element of the Hough space. The introduction of parallelism only in the pixels requires $N^2$ processors, one for each pixel. The number of CORDIC operations is $N/2$ + latency which depend on radix. There are also conflicts in the votation. A solution that does not produce voting conflicts is the parallelization of the angles. In this case a processor per angle is needed in which all the pixels of the image are procesed sequentially. The total number of processors is $N/2$ and the number of cycles for the evaluation of the transform is $N^2$ + latency and one pixel is processed in each cycle.

The implementation considered in [27] uses a 12-bits-precision CORDIC processor with 10 stages (6 are the standard iteration-stages, 1 for the compensation of the scaling factor and 3 for performing the scaling). Each stage consists of two registers, two multiplexers and two adders/subtracters. The standard stage needs 24 bits for each angle in the ROM.

To transform a 128·128 image with 128 rotation angles, in the case of angle paralellization, 64 processors are needed and 128·128 cycles are performed. The area of the n-bit register can be estimated as $0.5n\cdot\tau_a$. The area of the multiplexers depends on the number of input vectors $v$ and on their wordlength $n$. The associated area is about $0.25\cdot v\cdot n\cdot\tau_a$. The whole area estimates is shown in Table 6:

| Parallel CORDIC | Quant. | Complex gates |
|---|---|---|
| **Registers** | 20·64 | $20\cdot64\cdot0.5\cdot12\ \tau_a = 7880\cdot\tau_a$ |
| **Multiplexers** | 20·64 | $20\cdot64\cdot0.25\cdot2\cdot12\ \tau_a = 7880\cdot\tau_a$ |
| **Adders/subtracters** | 20·64 | $20\cdot64\cdot12\cdot\tau_a = 15760\ \tau_a$ |
| **LUT tables** | 64·24bits | $64\cdot24\cdot40\ \tau_a$/Kbit= $60\ \tau_a$ |
| **Overall** | - | $31580\tau_a$ |

**Table 6.** Area estimates for parallel CORDIC

Delay estimates must consider $0.5\tau_t$ for the multiplexers and $1\tau_t$ for the registers. Assuming $\tau_t$=1ns:

| Parallel CORDIC | Quant. | Complex gates |
|---|---|---|
| **Registers** | 10 | $128\cdot128\cdot10\cdot1\tau_t$ =0,164 ms |
| **Multiplexers** | 10 | $128\cdot128\cdot10\cdot0.5\ \tau_t$ = 0,082 ms |
| **Adders/subtracters** | 10 | $128\cdot128\cdot10\cdot\lg 12\ \tau_t \approx$ 0,573 ms |
| **LUT tables** | 64 | $64\cdot3\ \tau_t$ = 192 ns |
| **Overall** | - | 0.819 ms |

**Table 7.** Delay estimates for parallel CORDIC

CBRM parallelization consists in performing the calculation with $m$ CBRMs and each of them is charged of $N^2/m$ pixels. So, delay and area are to be modified by a factor $m$. Tables 8 and 9 show area and delay estimates for parallel CBRM, in the case of a 128·128 image, with 12 bits data precision. It can be noticed that factor $m$ doesn't affect the memory estimates because a single multiaccessed memory is used.

| Parallel CBRM | Impl.1 Complex gates | Impl.2 Complex gates |
|---|---|---|
| **k=1** | $7,5\ \tau_a + (12\cdot1\cdot\tau_a)m$ | $7,5\ \tau_a + (2\ \tau_a + 12\cdot1\tau_a)m$ = $7,5\ \tau_a + (14\tau_a)m$ |
| **k=2** | $15\ \tau_a + (6\cdot2\cdot\tau_a)m$ | $15\ \tau_a + (2\ \tau_a + 6\cdot2\tau_a)m$ = $15\ \tau_a + (14\ \tau_a)m$ |
| **k=4** | $105\ \tau_a + (4\cdot3\tau_a)m$ | $105\ \tau_a + (2\ \tau_a + 4\cdot3\tau_a)m$ = $105\ \tau_a + (14\ \tau_a)m$ |
| **k=8** | $102400\ \tau_a + (3\cdot4\cdot\tau_a)m$ | $102400\ \tau_a + (2\ \tau_a + 3\cdot4\tau_a)m$ = $102400\ \tau_a + (14\ \tau_a)m$ |

**Table 8.** Area estimates for parallel CBRM

| CBRM | Implementation 1 | Implementation 2 |
|---|---|---|
| **k=1** | $(12\cdot64\cdot128\cdot128\cdot(3\tau_t+\tau_t \lg 12))/m$ =81,789 ms/$m$ | $(12\cdot64\cdot128\cdot128\cdot(3\tau_t+2\tau_t)+\tau_t \lg 12)/m$=62,914 ms/$m$ |
| **k=2** | $(6\cdot64\cdot128\cdot128\cdot(3\tau_t+\tau_t \lg 6))/m$ =34,603 ms/$m$ | $(6\cdot64\cdot128\cdot128\cdot(3\tau_t+2\tau_t)+\tau_t \lg 6)/m$=31,457 ms/$m$ |
| **k=4** | $(3\cdot64\cdot128\cdot128\cdot(3.5\tau_t+\tau_t \lg 3))/m$ =15,729 ms/$m$ | $(3\cdot64\cdot128\cdot128\cdot(3.5\tau_t+2\tau_t)+\tau_t \lg 3)/m$ =17,301 ms/$m$ |
| **k=8** | $(2\cdot64.128\cdot128\cdot(5\tau_t+\tau_t \lg 2))/m$ =12,583 ms/$m$ | $(2\cdot64\cdot128\cdot128\cdot(5\tau_t+2\tau_t)+\tau_t \lg 2)/m$ =14,680 ms/$m$ |

**Table 9.** Delay estimates for parallel CBRM

It appears that in order to reach the same delay than parallel CORDIC, namely 0,819 ms, parallel CBRM needs different $m$ values. Table 10 shows the area estimates corresponding to these $m$ values.

| Parallel CBRM | Impl.1 Complex gates | Impl.2 Complex gates |
|---|---|---|
| **k=1** | $m$ = 100 $1207,5\tau_a$ | $m$ = 75 $1057,5\tau_a$ |
| **k=2** | $m$ = 43 $531\tau_a$ | $m$ = 39 $561\tau_a$ |
| **k=4** | $m$ = 19 $333\tau_a$ | $m$ = 21 $399\tau_a$ |
| **k=8** | $m$ = 16 $102592\tau_a$ | $m$ = 18 $102652\tau_a$ |

**Table 10**.Area estimates for parallel CBRM when delay is 0,819 ms

For the same time delay (0,819 ms) parallel CBRM needs less area up to $k$ = 4 than parallel CORDIC, as shown in Tables 6 and 10. If comparing the delay and area for the same number of processors, namely $m$=64, the obtained results for implementation1 of CBRM are: $k$=1 area=775,5 $\tau_a$ delay=1,278 ms; $k$=2 area=783 $\tau_a$ delay =0,540 ms; $k$=4 area=873 $\tau_a$ delay =0,246 ms; $k$=8 area=103168 $\tau_a$ delay =0,197 ms. We can observe that for $k$=2 and $k$=4, performances are better for both delay and area results in the case of parallel CBRM; however, for $k$=1 CBRM provides less area but longer delay and for $k$=8 more area and shorter delay.

The referred work [27] does not provide data on the error performed by the implementation presented. It suits to mention in all cases that the implementation of the CBRM can achieve even more speed when proposing four functions $\rho_I(k)$ $\rho_{II}(k)$ $\rho_{III}(k)$ $\rho_{IV}(k)$: the angle to rotate would be the half, and, for the same increment, the number of iterations would be divided by two. This improvement would provide additional time-saving by means of duplicating the hardware used.

## 7. Conclusions

A function evaluation method that provides calculation improvements for the HT has been presented (CBRM). It outlines the interest of convolution as a powerful tool that increases computational capabilities, essentially when applied to massive calculations. The evaluation of the HT has been carried out with a DA-based architecture which can be serial or by *k*-blocks, in order to provide computational improvements. Compared with other well-known proposals, namely pipelined and parallel CORDIC, it has been confirmed that the CBRM provides memory and hardware resource saving as well as speed improvements according to the experiments carried out with the HT. These encouraging partial conclusions make quite reasonable to study in depth the capabilities of convolution to provide a more complete set of recursive evaluating patterns in order to extend the CBRM.

## 8. References


[1] Muamar, H.K and Nixon, M., "Tristage Hough Transform for multiple ellipse extraction," *IEEE Proc. Part E: Computer and Digital Techniques,* Vol 138 nº 1, 1991.

[2] Haule, D.D and Malowany, A.S., "Object Recognition using fast adaptative HT," *IEEE Comp. Pacific Conf. On Communication, Compiler and Signal Processing,* pp. 91-94, 1989.

[3] da Silva, I., "Vectorization from aerial photographs applying the HT method," *Proc. SPIE, Vol 1395,* Pt2, pp. 956-963, 1990.

[4] Yamazawa, K.; Yagi, Y. and Yachida, M.: "3d Line Segment Reconstruction by Using Hyperomni Vision and Omnidirectional Hough Transforming," *ICPR00,* Vol III: 487-490, 2000.

[5] Bariani, M.; Cucchiara, R.; Mello, P. and Piccardi, M.: "Exploiting symbolic learning in visual inspection," *Proc. of IDA 97 4-6 1997, Lecture Notes in Computer Science*, v. 1280, Springer, pp. 223-234 (ISBN 3-540-63346-4), 1997.

[6] Dong, F.; Clapworthy, G.J. and Krokos, M. "Volume Rendering of Fine Details Within Medical Data," *IEEE Visualization*, San Diego, 2001.

[7] Tezmol, A.; Sari-Sarraf, H.; Mitra, S. and Gururajan A.: "Customized Hough Transform for Robust Segmentation of Cervical Vertebrae from X-Ray Images," *Fifth IEEE Southwest Symposium on Image Analysis and Interpretation,* April, Santa Fe, New Mexico, 2002.

[8] Huang, L.Y.; Hu, Z. and Sun, F.M.: "A New Automatic Quasar Recognition Technique Based on PCA and the Hough Transform," *ICPR 2000*, pp. 2499-2502, 2000.

[9] Sural S. and Das, P.K.: "A genetic algorithm for feature selection in a neuro-fuzzy OCR system", *Sixth International Conference on Document Analysis and Recognition*, pp. 987-991, Seattle, 2001.

[10] Koshimizu, H. and Numada, M., "On fast Hough Transform method PLHT based on piecewise-linear Hough function," *J. System Computer in Japan,* Vol 21 nº5, pp. 62-73, 1990.

[11] Ben-Tzvi, D. and Sandler, M.: "A Combinatorial Hough Transform". *J.P. Recognition Letters,* Vol 11, pp. 167-174, 1990

[12] da Fontura, L. and Sandler, M.B.: "A binary HT and its efficient implementation in a systolic array architecture," *J.P. Recognition Letters*, Vol. 10, pp. 329-334, 89.

[13] Walther, J.S.: "A unified algorithm for elementary functions," *Proc. Spring Joint Computers Conf.*, pp. 379-385, 1971.

[14] Li, H.F.; Lavin, M.A. and Le Master, R.J.: "Fast Hough Transform: a hierarchical approach," *J. Computer Vision Graphics Image Processing*, Vol.36, pp. 139-161, 1986.

[15] Hu, Y.H.: "CORDIC-based VLSI architectures for Digital Signal Processing," *IEEE Signal Processing Magazine*, nº 7 pp. 16-35, 1992.

[16] Shankar, R.V. and N. Asokan.: "A parallel implementation of the Hough Transform method to detect lines and curves in pictures," *IEEE 32th Midwest Symp. On Circuits and Systems*, pp. 321-324, 1990.

[17] Chuang ,H.Y.H. and Li, C.C. "A systolic processor for straight line detection by modified HT," *IEEE Conf. on Computer Architecture for Pattern Analysis and Image Database Management*, pp. 300-304, 1995.

[18] González, R.C; Woods, R.E.: "Tratamiento digital de imágenes," *Addison-Wesley. Iberamericana* S.A. 1996

[19] Katsuhiko O.: "Ingeniería de control moderna," *Prentice-Hall Hispanoamericana,* 1993.

[20] Serra, J.: "Image Analysis and Mathematical Morphology," *Academic Press, London,* 1982.

[21] Kittler, J. and Alkoot, F.M.: "Sum versus Vote Fusion in Multiple Classifier Systems," *IEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 25, nº 1, Jan. 2003.

[22] Weyl, H. and Meter, F.: "Die Vollstaändigkeit der primitiven Darstellungen einer geschlossen kontinuierlichen Gruppen". *Math. Ann. TXCVII*, pp.737-755, (1927).

[23] Ercegovac, M. and Lang, T.: "Division and Square root: Digit-Recurrence, Algorithms and Implementations", *Kluwer Academic Pub.*, 1994

[24] Piñeiro, J-A.; Bruguera, J.D. and Muller. J.M.: "Faithful powering computation using table look-up and a fused accumulation tree". *Proc. of the 15th International Symposium of Computer Arithmetic (ARITH'15),* 2001.

[25] Piñeiro, J-A.; Ercegovac, M.; Bruguera, J.D.: "High-Radix Logarithm with selection Rounding," *IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02)*, p-101, July 2002.

[26] Deng, Dixon, D.S and El Gindy H., "High speed Parametrizable HT using reconfigurable hardware," *Pan-Sydney Area Workshop and Visual Information Processing (VIP)*, 2001.

[27] Bruguera, J.D. and Guil, N.: "CORDIC-based parallel/pipelined architecture for the HT," *J. of VLSI Signal Processing*, Vol 12, nº 3, pp. 207-221, 1996.