Available online at www.sciencedirect.com

Image and Vision Computing 23 (2005) 1009–1017

www.elsevier.com/locate/imavis

Real-time FPGA implementation of Hough Transform using gradient and CORDIC algorithm

Si Mahmoud Karabernou*, Fayçal Terranti

ETIS—ENSEA/UCP UMR 8051 CNRS 6, Avenue du Ponceau, 95014 Cergy Pontoise, Cedex, France

Received 25 March 2004; received in revised form 20 May 2005; accepted 1 July 2005

Abstract

This article describes a real-time architecture for Hough Transform based on CORDIC algorithm and gradient for straight lines detection. The CORDIC algorithm allows the use of simple operators, such as adders and shift registers, whereas the use of gradients reduces considerably the computation quantity. The proposed real-time architecture can be easily fitted into Field Programmable Gate Array (FPGA) reconfigurable devices, since the present performance increase of this technology allows the implementation of complex applications while real-time constraints in Hough Transform for lines detection are respected for most of the video transmission standards. This approach can be easily applied for other shapes detection.
© 2005 Elsevier B.V. All rights reserved.

Keywords: CORDIC; FPGA; Hough; Real-time

1. Introduction

It is known that the Hough Transform (HT) is widely used in image processing and particularly in pattern recognition such as lines, circles, ellipses and other shapes. The Hough Transform principle is also used in 3D object recognition, rigid bodies orientation, motion parameters of mobile objects and etc. [1–3]. The Hough Transform is an attractive technique because of its robustness towards noise. However, its main disadvantages are the large memory size required and the necessary complex and time consuming computations.

A lot of work has been carried out in relation to real-time HT hardware implementation [4,5]. While the L64250 [6,7] chip of Logic Company uses a modified Hough Transform for a real-time processing. O'Gorman and Clowes [8] have proposed the use of the gradient in order to reduce computation's complexity. Additionally, Zhou and Kornerup [9] have proposed the use of the CORDIC algorithm for Fast Hough Transform. The proposed real-time architecture for straight lines detection implemented on FPGA devices, uses both the gradient and CORDIC algorithm. On one hand, the gradient have been used in order to reduce the amount of calculations so that less time is required, on the other hand the CORDIC algorithm has been used to make the operators less complex. Section 1 presents shortly the HT principle followed by a discussion related to different computation methods. The CORDIC algorithm is briefly described in Section 4. Details about some implementations of the CORDIC algorithm are shown in order to choose the appropriate structure for our architecture. Section 7 describes the straight lines detection algorithm and the proposed architecture. Section 8 presents the system prototype and finally the obtained results.

2. Hough Transform principle

The aim of the Hough Transform is to represent regular geometric forms in a parameter space defined by ρ and θ. If a straight line is considered, ρ represents the normal distance from the origin to the straight-line and θ represents the angle between the normal and the X-axis as shown in Fig. 1. A straight line, is therefore, represented by the Eq. (2) instead of the relation (1):

$$y = ax + b \tag{1}$$

$$\rho = x\cos\theta + y\sin\theta \tag{2}$$

* Corresponding author. Tel.: +33 1 30 73 66 18; fax: +33 1 30 73 66 27.
E-mail address: karabernou@ensea.fr (S.M. Karabernou).

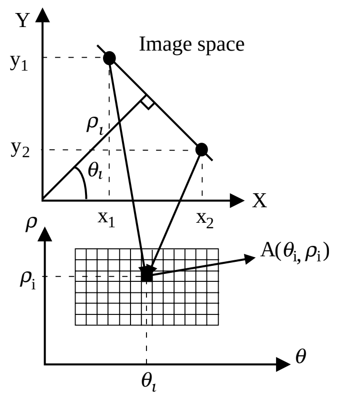

Fig. 1. Hough Transform principle.

In the parameter space, each straight line is represented by a single point as $L_k(\rho_k,\theta_k)$. According to Eq. (2), each point that belongs to a straight line, has its corresponding sinusoidal curve in the parameter space. All sinusoidal curves corresponding to a straight line points will cross through a single point in the parameter space (ρ_0, θ_0). In other words, the input object for the parameter space computation is a contour image. For each feature point in the contour image, the corresponding sinusoidal curve is computed using the relation (2). If a set of feature points in the contour image belongs to the same straight line, their corresponding sinusoidal curves will have a common point in the parameter space. This point is found out using a voting process and will constitute the straight line parameters as depicted in Fig. 1. Using the Hough Transform principle, the task of detecting straight lines in the (x, y) space, is reduced to finding out the sinusoidal curves intersection points in the parameter space. These crossing points are simply parameter memory space positions having high votes.

3. Hough Transform computation methods

There are two main approaches to compute the Hough parameter space. The first one is called classical method, where all possible values of θ_i parameter are used to calculate the corresponding ρ_i values for a single given feature point. This approach requires long computation time and a large memory space. The second method is based on the gradient orientation, where only a single value of θ is used to calculate the corresponding ρ_i value for a given feature point. The θ value is determined from the horizontal and vertical gradients at the considered feature point.

3.1. Classical methods

The procedure to compute the Hough parameter space using the classical method is summarized in the following algorithm:

Table 1
Complexity for classical Hough Transform

Additions	*C.K*
Products	2.*C.K*
Look Up Table access	2.*C.K*
Parameter space memory access	*C.K*

```
The parameter space is a matrix R(ρ, θ)
R(ρ, θ) is initialized to zero
For each feature point(x, y) in the contour image
Begin
    For each value $\theta_k \in [\theta_{min}, \theta_{Max}]$
    begin
        Compute $\rho_k = x.\cos\theta_k + y.\sin\theta_k$
        Let $R(\rho_k, \theta_k) = R(\rho_k, \theta_k) + 1$
    end
End
```

If the sine and cosine values are obtained from Look Up Tables (LUT). Table 1 summarizes the computation complexity required, where K represents all θ possible values and C the feature points number in the considered contour image.

3.2. Fast incremental Hough Transform FIHT2

FIHT2 stands for the Fast Incremental Hough Transform [10]. It is a modified HT that gives the same result with only simple operators such as adders and logical shifters.

This new version of HT computation is based on the fact that θ can be written as $\theta_n = (n.\pi)/K = n.\varepsilon$, where $n \in 0, 1, K-1$ and ε represents the quantification step of θ. The parameter space $R(\rho, \theta)$ may be redefined by the relations (3) and (4):

$$\rho_n = x\cos\theta_n + y\sin\theta_n + (p/2K)x\sin\theta_n \tag{3}$$

$$\rho'_n = -x\cos\theta_n + y\sin\theta_n - (p/2K)x\sin\theta_n \tag{4}$$

The trigonometric evaluations as well as the products may be avoided if $\varepsilon = p/K = 1 = 2^m$ with $K = p.2^m$ and m is a natural number. Eqs. (3) and (4) become:

$$\rho_{n+1} = \rho_n + \varepsilon\rho'_n \quad \text{for } 0 \le n < K/2 \tag{5}$$

$$\rho'_{n+1} = \rho'_n - \varepsilon\rho_{n+1} \quad \text{for } K/2 \le n < K \tag{6}$$

Since ε is a power of 2, the products are reduced to simple shift operations. This method needs that ρ_0 and ρ'_0, be respectively, initialized to the x and y values of the current feature point in order to start computations. For $m=6$, $K=201$ and according to Table 1, the 402.C products are reduced to 199.C logical shift operations. The introduced error using relations (5) and (6) with respect to Eq. (1) is about 0:002. Since this value is less than pixel resolution, FIHT2 technique can be used for straight line detection, if K is chosen sufficiently large. However, this method still needs $C.K$ additions and $C.K$ parameter space memory accesses because it uses all possible θ values.

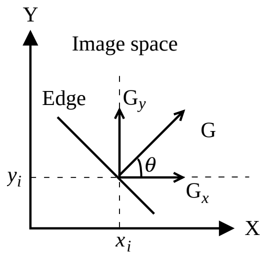

Fig. 2. Edge–Gradient orientation relationship.

3.3. Gradient-based methods

As described before, this approach uses the gradient to estimate the straight line orientation at the considered feature point. Since a single value of θ will be needed, only a unique memory access will be required for each feature point. This will, of course, accelerates computations and reduces memory size requirements. The weak quality of edge detectors did not allow a reliable usage when O'Gormans and M.B. Clowes have introduced this idea in 1976. The horizontal gradients G_x and vertical gradients G_y are obtained from preprocessing operators while extracting the contour image as Robert, Sobel, Prewitt, or using Deriche edge detector preceded by an image smoothing [11].

$$G = \tan\frac{|G_y|}{|G_x|} \quad \text{if } |Gy| \le |Gx|.$$
$$G = \tan\frac{|G_x|}{|G_y|} \quad \text{otherwise.} \tag{7}$$

As shown in Fig. 2, we aim to compute the slope of the gradient G at each feature point in the contour image. According to relation (7), G is always less than 1, which guarantees detection precision. It is obvious that the use of the gradient reduces considerably the memory size as well as memory access number. Fig. 3 shows the difference in size of the parameter spaces obtained by both classical and gradient-based methods. However, the number and the complexity of the operators are still important. CORDIC algorithm offers the possibility to avoid heavy operators, such as trigonometric calculations and dividers and use only additions and logical shifts.

4. CORDIC algorithm

COordinate Rotation DIgital Computer (CORDIC) algorithm was first described in 1959 by Jack E. Volder [12] as a numerical solution to trigonometric problems in real-time navigation's applications. Subsequent work carried out by Walther [13] extended the use of CORDIC to other functions. CORDIC algorithm has been used in several systems namely the 8087 math coprocessor, the HP-35 computer and other radar dedicated processors. Each trigonometric function can be computed using vector rotation. The algorithm provides an iterative method to perform vector rotation by an arbitrary angle ϕ using only additions, comparisons and shift operations. The Volder algorithm is derived from the general equation of the vector rotation.

If a vector $V(x, y)$ is rotated by an angle ϕ, the new vector $V'(x', y')$ can be expressed by relation (8) or (9)

$$x' = x\cos\phi - y\sin\phi \qquad y' = y\cos\phi + x\sin\phi \tag{8}$$

$$x' = \cos\phi[x - y\tan\phi] \qquad y' = \cos\phi[x + y\tan\phi] \tag{9}$$

These expressions are still complex for hardware implementation. The multiplication by $\tan\phi$ may be avoided if the rotation angle is limited to values such that $\tan\phi = 2^{-i}$. This will lead to simple shift operations. Arbitrary rotation angle may be obtained using several elementary rotations with $\phi = \arctan 2^{-i}$. Therefore, the above relations may be rewritten according to relations (10) and (11)

$$x_{i+1} = K_i[x_i - y_i d_i 2^{-i}] \tag{10}$$

$$y_{i+1} = K_i[y_i - x_i d_i 2^{-i}] \tag{11}$$

where

$$K_i = \cos(\arctan(2^{-i})) = 1/\sqrt{(1 + 2^{-2i})} \quad \text{and} \quad d_i = 1.$$

The product of all K_i values is called the K factor. Its value can be separately obtained and it equals to 0.6073 when n goes to infinity. Therefore, the algorithm has a gain $G_n \approx 1{:}647$. In order to simplify the hardware implementation, the approximation $\arctan(2^{-i}) \approx (2^{-i})$ for $i \ge 4$ may be performed. The $\arctan(2^{-i})$ values may be straightforwardly obtained from a small Look Up Table (one input per iteration).

Going with these bases, the CORDIC algorithm uses two modes for determining the elementary rotation sign: Rotation mode and Vector mode. In rotation mode, the accumulator angle is initialized to the desired angle z_0. For a given iteration, the rotation sign is chosen in such a way to minimize the residual angle in the accumulator. The decision, is therefore, based on the residual angle sign. After n iterations, Eqs. (12) or (13) are obtained.

$$x_n = A_n[x_0\cos(z_0) - y_0\sin(z_0)] \tag{12}$$

$$y_n = A_n[y_0\cos(z_0) + x_0\sin(z_0)] \tag{13}$$

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

In vector mode, the CORDIC algorithm rotates the input vector $V(x_0, y0)$ by the appropriate angle so that the resulting angle will be aligned with the X-axis. The result is hence a rotation angle (angle accumulator Z) and the input vector is multiplied by the gain G_n (X component of the r result). This mode attempts to minimize the residual vector Y component. The next sign rotation is determined according to

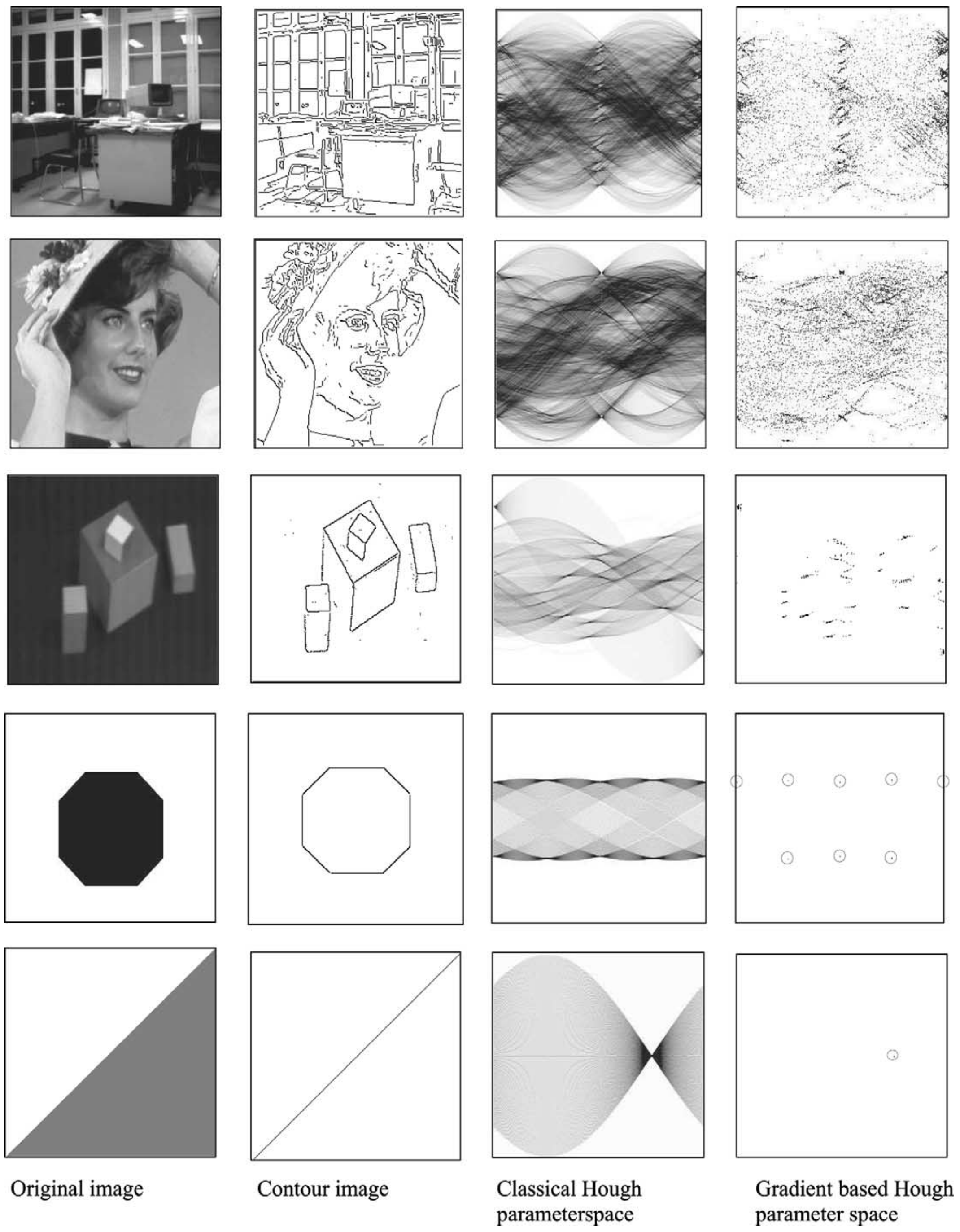
Original image
Contour image
Classical Hough parameterspace
Gradient based Hough parameter space

Fig. 3. Classical and gradient-based Hough Transform parameter space size comparison.

the last Y component sign. After n iterations, x_n, z_n and y_n may obtained by relation (14) or (15)

$$x_n = A_n\sqrt{x_0^2 + y_0^2} \tag{14}$$

$$z_n = z_0 + \arctan(y_0/x_0) \quad y_n = 0 \tag{15}$$

In both modes, the rotation angle is limited to values between $-\pi/2$ and $+\pi/2$. This limitation is due to that the value 2^0 has to be used for the arc tangent at the first iteration. For angles greater than $\pi/2$, an initial rotation of $\pm\pi/2$ will lead to the new equations expressed in (16).

$$x' = -\mathrm{d}y \quad y' = \mathrm{d}x \quad z' = z + \mathrm{d}\pi/2 \tag{16}$$

where $d = +1$ if $y < 0$ and -1 if $y \geq 0$.

From the above descriptions, the CORDIC algorithm allows direct calculation of several trigonometric functions as well as the computation of other operations in an indirect way if an appropriate choice of initial values and rotation modes are performed [14].

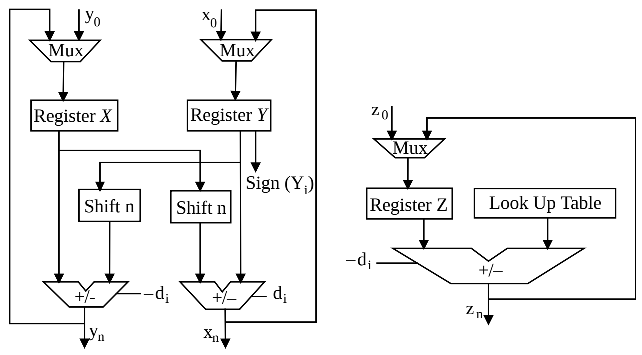

Fig. 4. CORDIC Parallel iterative architecture.

5. CORDIC architectures for FPGA implementations

The previous modified CORDIC equations may be implemented in different manners: serial, parallel, iterative or cascade. Several choices are possible. The mostly used architectures are well described in Ref. [15,16].

In parallel iterative implementation, the least significant bit of Y (or Z according to the mode used) register determines whether an addition or a sub-straction operation has to be performed for the next iteration. A simple state machine is necessary to control the different elements and sequence the algorithm. This architecture is shown in Fig. 4.

In parallel-cascaded form-based implementation of CORDIC equations, the parallel structure of a single bit is cascaded as much as the number of iterations and the number of bits. This is shown in Fig. 5.

Two simplifications are made when cascaded structure is used. The first one, only 1 bit shift operators are needed instead of n bits shift operators. The second one, the Look Up Table is now distributed over all the structure. Also the (x_0, y_0, z_0) values are input straightforwardly without multiplexing. It is obvious that FPGA devices are not suitable for this kind of architectures. This is due to the large

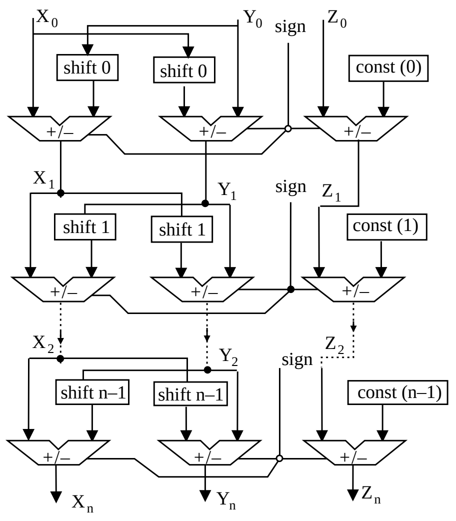

Fig. 5. CORDIC Parallel cascade architecture.

Table 2
The iteration number effect on complexity and frequency

Iteration number	Complexity (CLBs)	Max. frequency (MHz)
8	184	6.10
9	208	5.64
10	232	4.83
11	256	4.43
12	280	3.78
13	304	3.48

number of interconnections required at each stage. When the processing is done on 16 bits wide words, each stage will require 48 inputs and outputs. Depending on CAD tools available, generally this results to a very hard FPGA placement and routing process and obviously, additional delays are introduced. This is true for small FPGA devices.

The effect of iteration number on complexity and operational frequency can be clearly seen on Table 2 when the structure is implemented on a Xilinx 4010 device. The complexity is measured using Cellular Logic Block (CLB). For this device family particularly, a CLB includes 2 flip–flops, 2 Look Up Tables for combinational logic implementation and some other control logic. The fact that each CLB element includes two flip–flops, pipelining the structure will cost almost nothing. However, the maximum frequency is obtained when a register is added at every iteration. This can be seen on Table 3, which summarizes implementation reports with different pipeline register number. The frequency gain is much more important than the increase in CLB count. This architecture is very attractive when using high-density FLAG devices such as Virtex of Xilinx or Flex of Altera.

High complexity, long delay and the huge number of the required interconnections between elements within a single iteration stage, are the main disadvantages of parallel implementations. When a serial implementation is considered, the complexity is expected to be lower. However, the throughput and latency have to be closely studied. The throughput is defined with Eq. (17).

$$F/(N_iW) \tag{17}$$

Where F stands for the clock frequency, N_i for iteration number and W for the data word width. As shown in Fig. 6, an iteration requires W cycles.

Table 3
The pipeline register effect on complexity and frequency

Iteration number between pipeline registers	(CLBs)	Max. frequency (MHz)
1	313	24.4
2	208	18.3
3	304	14.2
4	304	9.7
8	304	6.2
13	304	3.7

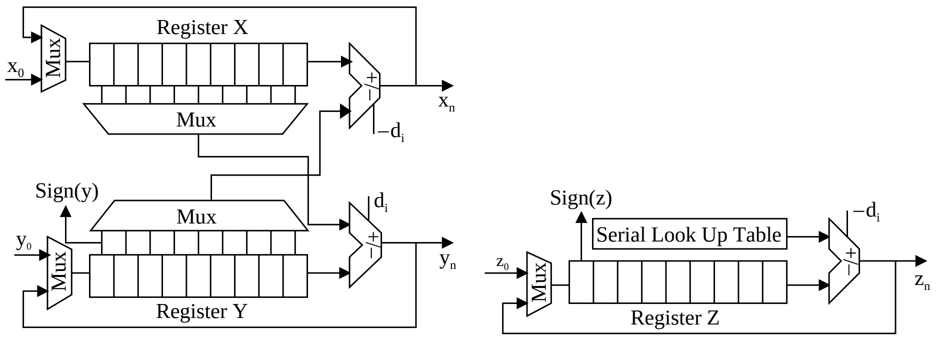

Fig. 6. CORDIC series architecture.

This architecture operates with frequencies higher than that of parallel architectures and can be easily fitted into a single FPGA device. Due to the architecture simplicity, a high operational frequency could be reached. Also, the throughput could be similar to parallel cascaded structure. Table 4 summarizes the report of the iterative serial implementation with 13 iterations on the Xilinx 4010 FPGA device.

6. Real-time Hough Transform using CORDIC

In order to meet real-time constraints, it is important to define the target of the image/video specification [18]. When conventional TV and some MPEG standards are considered, the proposed architecture must be able to reach a processing rate of 25 frames per second. The processing should include parameter space calculations as well as straight-lines image building 720×576 pixels frame must be processed within 40 ms. Therefore, the processing of one pixel should not exceed 88.25 ns (a minimum frequency of 11.3 MHz). The proposed global architecture is shown in Fig. 7. Two CORDIC units are used to simultaneously compute ρ and θ. The two units use different modes. This first block is shown in Fig. 8. Details of the design of each unit are illustrated in Figs. 9 and 10.

In the global architecture shown in Fig. 7, the idea behind using two memories is to make, the space parameter calculations and straight lines frame rebuilding process, to be performed in parallel. While RAM1 is used for current image vote storage, RAM2, which contains the space parameter of the previous frame, is scanned to build the straight lines image frame. The two 11 memories are swapped at the end of each frame. The address bus of the parameter space memory is obviously the concatenation of ρ and θ.

While a memory is scanned, it is important to notice that at the end of each read cycle, the current memory position is cleared using the read-modify cycle principle. The reason of doing that is to cleanup the memory for the next frame.

When the data format of 9 bits for ρ and 12 bits for θ with nine iterations for CORDIC algorithm are used, the implementation concerning the parameter space calculation on a Xilinx XC4010EPC84 [17] chip, showed that twice the minimum frequency required may be used. The implementation results can be viewed in Table 5.

However, real-time constraints are met for only some of the image transmission standards. Table 6, where resolution is given in pixels and fps stands for frame per second, shows standards that can use this approach (gradient and CORDIC) to compute Hough Transform in real-time.

7. Straight line detection

The straight-lines detection block uses the parameter space memory as a starting object. The parameter space

Table 4
The serial iterative CORDIC FPGA implementation report

	Serial iterative
CLBs	111
Look Up Tables	153
Flip–flops	108
Frequency (MHz)	48
Latency (s)	5.33
Max. throughput	0.1875

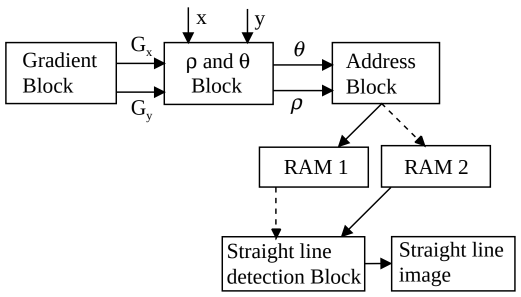

Fig. 7. Straight line detection global architecture.

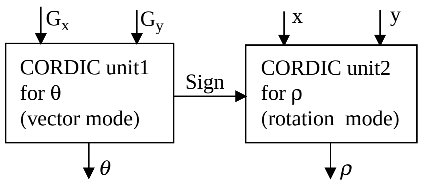
G_x
G_y
x
y
CORDIC unit1 for θ (vector mode)
Sign
CORDIC unit2 for ρ (rotation mode)
θ
ρ

Fig. 8. Gradient–CORDIC based Hough transform global architecture.

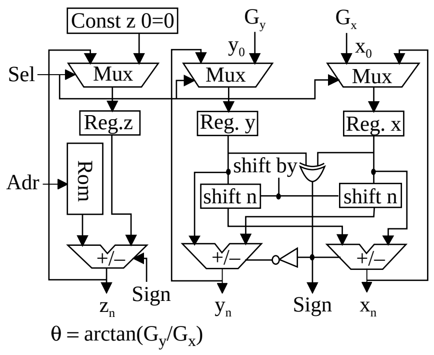

Fig. 9. θ parameter unit architecture.

memory has to be scanned to find out high vote values that define line's parameters. Some vote values may be small due to either short segments or vote dispersion. Insufficient precision and bad edge detector filter's quality is the main reasons for vote dispersion. In order to overcome this problem, the straight-lines detection block must include a threshold unit in which the minimum vote value is fixed to define a valid line. The threshold unit may be implemented with a simple comparator. The straight-lines detection block is based on the following algorithm:

```
Straight line image $(x, y)$ is initialized to zero
For each non-zero vote $(\rho, \theta)$ in parameter space
If vote $(\rho, \theta) \geq$ threshold
If $\theta \in [-\pi/4, \pi/4]$
For each $y=0$ to $y_{\max}$
$X = (\rho/\cos\theta) + x_{\max}/2(\tan\theta) + y_{\max} - y(\tan\theta)$
Increment line $(y, x)$ if $x \in [0, x_{\max}]$
Else
For $x=0$ to $x_{max}$
$Y = (\rho/\sin\theta) + x_{\max}/2(\cot\theta) + y_{\max}/2 - x(\cot\theta)$
Increment line $(y, x)$ if $y \in [0, y_{\max}]$
End
End
```

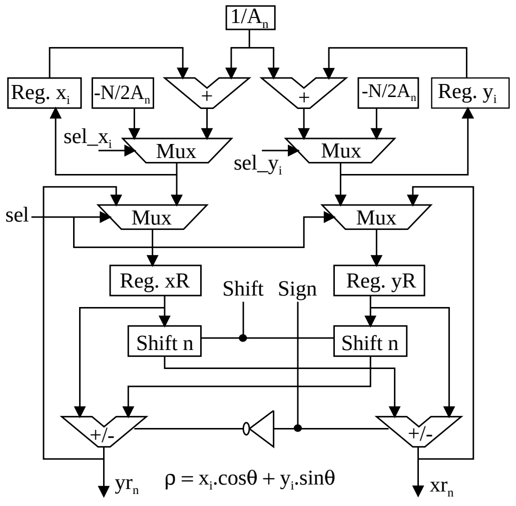

Fig. 10. ρ parameter unit architecture.

Table 5
Parameter space implementation report

Number of CLBs	205 out of 400	51%
4 inputs Look Up Tables	382 out of 800	47%
4 inputs Look Up Tables	46 out of 400	11%
Total CLB flip–flops	120 out of 800	15%
Minimum period	41.167 ns	
Maximum Frequency	23.166 MHz	

The algorithm is based on the general Hough Transform relation (2). In this process, the unknown variables are X and Y. Depending on the ρ value, whilst the parameter space memory is scanned, either x is a function of y or y is a function of x. As shown in Fig. 11, four CORDIC units and two adders are required. A first unit is designed to obtain both the $\sin\theta$ and $\cos\theta$ values. Two CORDIC units are designed to perform the two divisions ($\sin\theta/\cos\theta$ or $\cos\theta/\sin\theta$) and ($\rho/\cos\theta$ or $\rho/\sin\theta$). The last unit computes either $(y_{\max}-y).(\tan\theta)$ or $(y_{\max}/2-x).(\cot\theta)$.

Three CORDIC units are cascaded while the fourth one operates in parallel with the first unit. According to Table 5, and real-time constraint's definitions, the maximum time to treat a single pixel is not exceeded. This will guarantees a real-time run.

Fig. 12 shows the results of the above algorithm, implemented in C language, applied to different images.

Table 6
Image transmission standards that can use the proposed architecture for real-time Hough Transform parameter space calculations

Application	Resolution	Fps	
Slow motion video	176×120	10	Yes
Video conference	352×240	15	Yes
Video file transfer	352×240	15	Yes
Digital video (CD-ROM-based)	352×240	30	Yes
Broadcast video NTSC TV	720×480	30	Yes
Broadcast video PAL—SECAM	720×576	25	Yes
CIF format (luminance frame)	352×288	30	Yes
QCIF format	176×144	30	Yes
HDTV (High definition TV)	1280×720	≈60	No
HL MPEG-2 specification	1920×1080	30	No
HL MPEG-2 specification	1920×1152	25	No
H1140 MPEG-2 specification	1440×1080	30	No
H1140 MPEG-2 specification	1440×1152	25	No
ML MPEG-2 specification	720×480	≈30	Yes
ML MPEG-2 specification	720×576	25	Yes
LL MPEG-2 specification	352×288	≈30	Yes
Super high definition images	2000×2000	>60	No

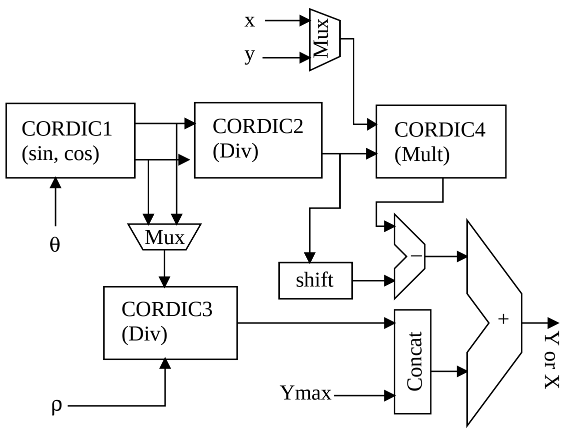

Fig. 11. Straight line rebuilding block architecture.

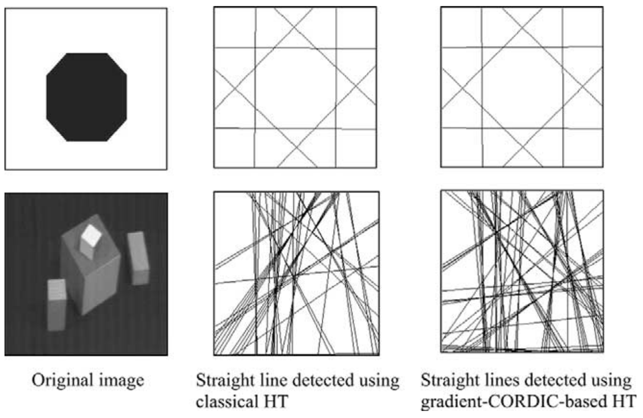
Original image
Straight line detected using classical HT
Straight lines detected using gradient-CORDIC-based HT

Fig. 12. Straight lines detection results.

The results are the same for simple images, however, images with more objects, additional lines are detected. This is due mainly to vote dispersion in the parameter space. Including a local maxima extraction unit to reject dispersed votes that are higher than the fixed threshold, should enhance the quality.

8. The prototype

The implementation of the above architectures has been carried out on the XC4010EPC84 in order to measure the computation time and evaluate the complexity. The whole design (edge detector, parameter space calculation and straight-lines image building) is to be implemented on the system shown in Fig. 13. The system includes a commercial camera, a data input–output stand-alone board from Celoxica, a Virtex PCI board also from Celoxica and finally a standard display screen. The stand-alone data input–output extracts pixel's bus and synchronization signals from the camera data flow in a numerical form. The PCI board includes a XILINX VIRTEX1000 FPGA chip, 8 MB static memory and the PCI bridge. The PCI bridge is used only to transfer the configuration bit file into the Virtex chip. Once processed on the VIRTEX1000 board, the pixel's data are got back on the data input–output board to be viewed on the standard display. With the VIRTEX1000 technology, designs may use system clock rates up to 200 MHz including input–output [19].

9. Conclusion

A real-time architecture for the gradient based Hough Transform has been proposed and developed. In order to implement the trigonometric functions and arithmetic operators in a suitable manner, the CORDIC algorithm has been used. Only adders and shift registers have been involved. Satisfactory results have been obtained from silicon area and speed point of view. The choice of nine iterations gave good results. Increasing the iteration number

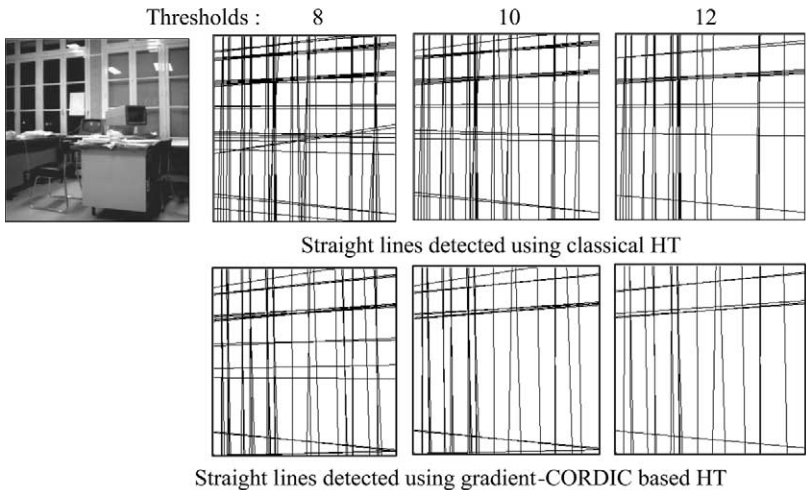
Thresholds : 8 10 12
Straight lines detected using classical HT
Straight lines detected using gradient-CORDIC based HT

Fig. 13. System prototype.

will enhance accuracy, but will slow down the processing rate. This work has shown that Hough parameter space calculations based on the gradient and CORDIC algorithm for lines detection may be implemented on medium scale FPGA devices, where real-time constraints are met for some of the image transmission standards. This approach will guide future work towards the use of additional iterations in the CORDIC algorithm, more bits for data representation and more complex shapes and objects can be extracted within the real-time constraints.

References

[1] P.V.C. Hough, Method and Means for Recognizing Complex Patterns, United State Patent. No. 3 069654, December, 1962.

[2] R.O. Duda, P.E. Hart, Use of the hough transform to detect lines and curves in pictures, CACM 15 (1) (1972) 11–15.

[3] H. Maître, Un Panorama de la Transformée de Hough, Traitement du Signal 2 (4) (1985).

[4] M. Meribout, M. Nakanishi, T. Ogur, A real-time image segmentation on massively parallel architecture, Real-Time Imaging 5 (1999) 279–291.

[5] D. Houzet, Video Rate Hough Transform Implementation on the SIMD/MIMD Parallel Architecture, GFLOP, Manuscript of July 15th 1993. Supported by CNRS PRC-GDR ANM.

[6] K. Hanahara, et al., A real-time processor for the Hough transform, IEEE Transactions on Pattern Analysis and Machine Intelligence 10 (1) (1988) 124–125.

[7] LSI Logic, *L64250 Histogram/Hough Transform Processor*, August 1989.

[8] F. O'Gormans, M.B. Clowes, Finding picture edges through collinearity of feature points, IEEE Transactions on Computers C-25 (4) (1976) 449–456.

[9] F. Zhou, P. Kornerup, A high speed Hough transform using CORDIC, Paper Presented at DSP95 Conference of Limassol, Chypre, June, 1995.

[10] H. Koshimizu, M. Numada, FIHT2 algorithm: a fast incrmental Hough transform, IEICE Transactions E 74 (10) (1991).

[11] D. Demigny, B. Mazar, *Transformée de Hough, détection de segments temps réel test pour une coopération FPGA/DSP*, Paper Presented at GDR/PRC ISIS, January, 1998.

[12] The CORDIC trigonometric computing technique, IRE Transactions on Electronic Computers EC 8 (3) (1959) 330–334.

[13] J.S. Walther, A unified algorithm for elementary functions, Spring Joint computer Conference Proceedings 38 (1971) 379–385.

[14] Y.H. Hu, The quantization effect of the CORDIC algorithm, IEEE Transactions on Signal Processing 40 (4) (1992) 834–844.

[15] N. Lindlbauer, Application of FPGA's to musical gesture and processing, Master Thesis, Department of computer science and Electrical of Landshut University Germany, November, 1999.

[16] R. Andrak, A survey algorithm for FPGA based computers, In Proceeding of the 1998 ACM/ SIGDA Sixth International Symposium on Field Programmable Gate Arrays, Monterey, CA, pp. 191–200, February 22–24, 1998. (http://www.andraka.com/biblio.htm).

[17] APS-X84 VHDL/FPGA Synthesis tutorial, FPGA Basics, http://www.users.erols.com/aaps/x84lab/FPGA.htm.

[18] M.G. Albanesi, M. Ferretti, D. Rizzo, Benchmarking Hough transform architectures for real-time, Real-Time Imaging 6 (2000) 155–172.

[19] Virtex data sheet http://www.xilinx.com/products/.