A Log-polar Image Processing System on a Chip

Fernando Pardo, Jose A. Boluda and Julio C. Sosa
Departament d'Informàtica
Universitat de València
Avda. Vicent Andrés Estellés S/N
46100 Burjassot, Spain
Email: Fernando.Pardo@uv.es, Jose.A.Boluda@uv.es, jucesosa@alumni.uv.es

Abstract—**This paper deals with the implementation, in a high density reprogrammable device, of a complete log-polar image processing system. The log-polar vision reduces the amount of data to be processed and simplifies several vision algorithms, making it possible the implementation of a complete processing system on a single chip. The image processing system has a conversion module from cartesian to log-polar coordinates that has been developed by means of the CORDIC algorithm, and a processing module for implementing differential algorithms as a pipeline of differentiation stages. A reconfigurable approach on a high-density chip combines software flexibility and hardware performance appearing as specially suited for systems with hardware restrictions. Moreover, the log-polar data reduction allows the use of the internal circuit RAM modules as intermediate frame-grabbers. Two algorithms have been synthesized into the reconfigurable device showing its flexibility.**

I. Introduction

The increasing density and performance of reconfigurable devices, and the device families of system-on-a-programmable chip (SOPC), make a reconfigurable approach as specially suited for systems with hardware restrictions. In this way, it is desirable hardware performance combined with software flexibility and low power consumption, weight and size for, as an example, an autonomous robot.

Space-variant vision emerges as an interesting image representation, since information reduction is interesting for a system with hardware restrictions. Specially the log-polar mapping shows, as a particular case of space-variant vision, interesting properties in addition to the selective reduction of information. The most remarkable mathematical property of log-polar images includes the simplification of image rotation and scaling along the optical axis.

The data reduction achieved by the log-polar transformation makes it possible the integration in a SOPC of a complete image processing system, since the frame grabbers employed are reduced to simple small internal RAM modules, and the image processing logic is small enough for its inclusion in the chip. Moreover, the chip reconfigurability is used for fine-tuning the log-polar transformation parameters and for choosing among several different image processing algorithms that can be synthesized into the SOPC.

II. Log-polar mapping

Fig. 1 shows the transformation between the camera focal plane (left) and the cortical or log-polar representation of an image (right). The cortical plane is divided in two areas:

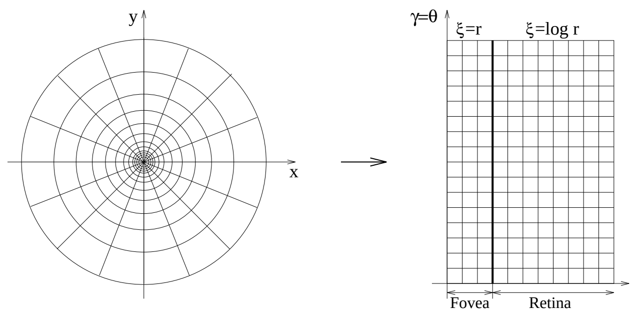

Fig. 1. Log-polar mapping from focal to cortical plane

the fovea and the retina. The fovea is the center part of the image and the retina is the periphery. It is necessary to make this distinction, since the log-polar transformation cannot be physically implemented due to the singularity of the logarithmic function at cero (origin); the density of pixels just in the image center is infinitum. For this reason, the image is divided in these two areas: in the retina the transformation is log-polar while in the fovea it is just polar to have a finite number of pixels in the center. There are other solutions for the fovea, but this one is interesting from the image processing point of view, since it provides a single processing plane.

The following equations define the retina and fovea transformation shown in Fig. 1:

$$\xi = \begin{cases} \xi_f \frac{r}{A} & \text{if } r < A \text{ (fovea)} \\ \xi_f + \frac{1}{B}\log\frac{r}{A} & \text{if } r \geq A \text{ (retina)} \end{cases} \qquad (1)$$
$$\gamma = \theta$$

where ξ_f is the frontier between the retina and fovea (this is to say, the number of rings in the fovea) and (r, θ) are the polar coordinates calculated from cartesian coordinates as follows:

$$r = \sqrt{x^2+y^2}$$
$$\theta = \arctan\frac{y}{x} \qquad (2)$$

where (x, y) are the cartesian coordinates of the pixel being transformed. The constants A and B of (1), tune the transformation between cartesian and log-polar: the A constant is the radius of the fovea measured in cartesian pixels, while B is an exponential growing factor, which value is usually calculated from A and the size of the cartesian image, so that the log-polar view field matches that of the cartesian image.

A. Transformation circuits

There are several solutions for obtaining log-polar images. The simplest solution consists of transforming the typical cartesian image obtained from almost any standard camera, as part of the image processing software. This solution is simple but it takes CPU time that could be better employed in other tasks, specially thinking on real-time image processing applications. The log-polar image generated using this approach has also the problem of the lack of matching between the cartesian and resulting log-polar image pixels, specially in the center of the log-polar image where resolution is usually higher than the cartesian image.

Other solution for the log-polar image acquisition is the use of a custom log-polar camera based on a log-polar image sensor [2]. This is the best solution for real-time applications, but it has also some problems. For example, these log-polar cameras are not widely available and they are not as cheap as standard cameras. In the other hand, the image quality obtained from log-polar sensors, though currently high, is not as good as most standard cameras.

The proposed solution is good for real time problems and offers good enough image quality for most applications. It is based on the LOG-CORDIC circuit that is a custom implementation of the cartesian to log-polar image transformation based on the CORDIC algorithm. This approach has the advantage of online reconfigurability, for log-polar mapping tuning, and high speed processing, taking no time of system CPU. The main problem of this approach is the same as the one of the software mapping, and it is the error introduced when transforming squared to log-polar pixels. For most applications this error is negligible and has little impact on the final processing results.

A circuit that performs a cartesian to log-polar image transformation can be implemented using different strategies. The simplest strategy consists of a look-up-table (LUT) for the coordinate transformation [3]. In this case, the hardware for the log-polar image transformation consist of the look-up-table, implemented on RAM or ROM, and a custom circuit for image transformation. The main problem with this approach is the look-up-table itself. Cartesian images are about 512×512 in size, which means a LUT of 256 K entries. The results of the LUT are the log-polar coordinates, around 76×128, which means two bytes for each entry of the LUT, giving a total LUT size of 512 Kbytes. This LUT size can be reduced taking into account the cartesian and log-polar symmetries, but it is still necessary to have a LUT of at least 128 Kbytes. Furthermore, the LUT capacity imposes a limitation when trying to transform larger cartesian images.

The finally implemented possibility, consists of calculating the transformation from cartesian to log-polar coordinates for every pixel coming from the camera. The advantages of this approach are the compact circuitry and the possibility of implementation as a part of a SOPC. The chosen implementation for the log-polar transformation has two stages: the first calculates the polar coordinates (radius and angle) of the cartesian coordinates (x, y) unsing CORDIC. The second stage then calculates the logarithm of the radius giving the final log-polar coordinates.

B. Computing log-polar coordinates using the CORDIC theory

CORDIC (COordinate Rotation DIgital Computer) is an iterative algorithm for the calculation of a rotation of a two dimensional vector in linear, circular or hyperbolic coordinate systems. It was first introduced by Volder [4] in 1959 and later generalized by Walther [5]. The idea behind this algorithm is to perform a rotation using simple shift and addition operations. It is possible to extend the rotation operation to obtain trigonometric operations as sine and cosine and even more complex as cartesian to polar transformations [6]. This algorithm can be further extended to hyperbolic functions, thus more complex operations as logarithms can be performed.

The simplest CORDIC algorithm is enough for our purposes and it is the one explained here. There are other CORDIC algorithms which may have higher performance [7]. Let (x_0, y_0) be the cartesian coordinates of a point and (x_n, y_n) the point after rotation of a θ angle. The equation for this transformation is as follows:

$$\begin{aligned} x_n &= x_0 \cos\theta - y_0 \sin\theta \\ y_n &= y_0 \cos\theta + x_0 \sin\theta \end{aligned} \tag{3}$$

This equation can be transformed to the following more convenient form:

$$\begin{aligned} x_n &= \cos\theta(x_0 - y_0 \tan\theta) \\ y_n &= \cos\theta(y_0 + x_0 \tan\theta) \end{aligned} \tag{4}$$

If the rotation angles are restricted so that $\tan\theta = 1/2^i$ $(i = 0, 1, \ldots, n)$, the multiplication by the tangent term is simplified to a simple shift operation. Taking this property into account, it is possible to decompose any rotation in a series of several, each time smaller, rotations which follow this restriction. In that case, any angle rotation operation can be performed by a series of shift and addition operations and a final multiplication (due to the $\cos\theta$ factor). The i stage of the series follows this equation:

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

where:

$$\begin{aligned} K_i &= \cos\theta_i = \cos\left(\frac{1}{\arctan^{-1} 2^{-i}}\right) = \frac{1}{\sqrt{1 + 2^{-2i}}} \\ d_i &= \pm 1 \end{aligned} \tag{6}$$

All K_i factors are constants that can be applied at the end of the rotation operation. This final product term can be considered as a gain of the circuit. When this CORDIC operation is part of a larger circuit, it is possible to include this multiplicative factor in other following operations for simplification. This is especially true for the LOG-CORDIC transformation circuit, where this factor has been completely cancelled, as it will be shown later. The gain factor A_n is the product of the inverse of all K_i:

$$A_n = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}} \tag{7}$$

The factor d_i, shown in (5) and (6), fixes the direction of the corresponding elementary i rotation. This sign on each stage depends on the angle to be shifted, and has the goal of approaching the result to the global angle at each stage. One convenient way to safely converge to the final angle, consists of defining a difference variable which informs about the proximity of the target angle. This variable is also useful to define the value for d_i:

$$\begin{aligned} z_{i+1} &= z_i - d_i \arctan 2^{-i} \\ d_i &= \begin{cases} -1 & \text{if } z_i < 0, \\ +1 & \text{if } z_i \geq 0 \end{cases} \end{aligned} \tag{8}$$

Equations (8) and (5), without the K_i constants, conform the CORDIC equations in rotation mode. The only operations needed to make any calculation are additions and shifts, since the $\arctan 2^{-i}$ is previously calculated and stored in a small table as part of the CORDIC circuit. These equations are useful for the computation of trigonometric functions. Given (x_0, y_0, z_0) the initial values, the CORDIC rotator in rotation mode gives the following results:

$$\begin{aligned} x_n &= A_n(x_0 \cos z_0 - y_0 \sin z_0) \\ y_n &= A_n(y_0 \cos z_0 + x_0 \sin z_0) \\ z_n &= 0 \end{aligned} \tag{9}$$

The CORDIC algorithm in rotation mode solves sine and cosine operations just giving the angle to z_0 and the right values to x_0 and y_0. For example, the sine is calculated making $x_0 = 1$ and $y_0 = 0$, the result is given in y_n.

There is another common mode for the CORDIC rotator and it is the vector mode. This mode consists of minimizing the y_n term instead of the z_n angle. This simple change produces a rotation until the rotated point reaches the x axis; the rotated angle is θ and the resulting x_n is r in polar coordinates:

$$\begin{aligned} x_n &= A_n\sqrt{x_0^2 + y_0^2} && = r \\ y_n &= 0 \\ z_n &= z_0 + \arctan\left(\frac{y_0}{x_0}\right) && = z_0 + \theta \end{aligned} \tag{10}$$

The equation for the d_i sign in vector mode must minimize y_i. The definition for d_i is then:

$$d_i = \begin{cases} -1 & \text{if } y_i < 0, \\ +1 & \text{if } y_i \geq 0 \end{cases} \tag{11}$$

This vector mode of the CORDIC rotator has been implemented as part of the cartesian to log-polar transformation circuit. The (x, y) coordinates of the image pixel are entered in the CORDIC circuit as (x_0, y_0) while $z_0 = 0$. In this way the results are directly the (r, θ) polar coordinates.

III. Processing log-polar images with differential algorithms

Differential algorithms, developed in log-polar coordinates, extract dynamic information using the temporal and spatial derivatives of the image sequence. These algorithms are computationally intensive due to the image size, but they benefit from log-polar data reduction. In this way, the implemented algorithms into the processing stage of the reconfigurable board, optimize temporal and spatial differential computations using double-port memories. The small log-polar image size allows the frame-grabbers integration in a single chip. In this way, the implemented algorithms into the reconfigurable circuit must optimize temporal and spatial differential computations. Initially two different algorithms have been proposed as processing stage.

A. Motion detection independent of the log-polar camera movement

Originally developed in Cartesian coordinates [8] and adapted to log-polar coordinates with a proved experimental effectiveness [9], this algorithm detects moving objects with respect to the static background. In a moving platform, there are image variations due to the self camera movement that may appear as moving objects with respect to the background. This algorithm is able to filter the image displacement due to the camera self movement. The constrains are related to several smoothness conditions in the grey level image and to the camera ego-motion. Theoretically, only objects which are moving with respect to the background are detected. The algorithm constrains are related to grey level and movement smoothness, and can be formulated as follows:

$$\frac{\partial^2 E(\xi, \gamma, t)}{\partial \xi^2} = \frac{\partial^2 E(\xi, \gamma, t)}{\partial \gamma^2} = \frac{\partial^2 \xi}{\partial t^2} = \frac{\partial^2 \gamma}{\partial t^2} = 0 \tag{12}$$

For any point of the cortical plane the grey level image must be smoothed, and the camera movement must be linear along the optical axis. Te focus of expansion must be in the center of the sensor. This movement is transformed in a translation along the radial coordinate. Under these constrains, the second temporal derivative of the image vanishes everywhere except for the self-moving object. Therefore, the algorithm is summarized as follows: First, the image must be smoothed, next the first order temporal derivative must be calculated from two consecutive images, and finally the second temporal derivative must be computed, selecting the zero values to binarize the image and marking self-moving objects. In this way the algorithm implementation into the programmable device must compute efficiently image temporal differences. In fact, due to the non-exactly accomplishment of the algorithm conditions, the condition for a point that belongs to a self-motion object is that the second temporal derivative of the log-polar images must be larger than a threshold.

B. Time to impact computation

A second differential algorithm based on log-polar vision is the time to impact computation of a camera to an approaching surface [10]. The time to impact (τ) can be computed in the sensor plane. In the case of polar images and supposing an approaching movement along the optical axis at the sensor center, this magnitude is:

$$\tau = K \frac{-\frac{\partial E}{\partial \xi}}{\frac{\partial E}{\partial t}} \tag{13}$$

where K is a constant that depends on the log-polar transform performed, $E(\xi, \gamma, t)$ is the log-polar image sequence, $\frac{\partial E}{\partial \xi}$ is the radial gradient and $\frac{\partial E}{\partial t}$ is the first order temporal derivative. Equation (13) shows that the time to impact can be computed as a division of two differential magnitudes. A previous stage for smoothing the original log-polar images is required for avoiding non-sense derivatives, like derivatives at the edges.

IV. Overall system synthesis

The overall system has been developed into an APEX PCI board from Altera that includes a SOPC APEX 20KC device (EP20K1000C) which has 38.400 Logic Elements (LEs) equivalent to 10^6 gates (or $1.7 \cdot 10^6$ system gates) and 320 Kbits of RAM. Furthermore, the board follows the mechanical and electrical PCI interface specifications and it is designed for the integration of a PCI mega-core as input/output interface. A 64 bits, 66 MHz master PCI interface fills around 1.400 LEs, which is less than 4% of the resources. This board and this device has been employed for synthesizing the image processing module that will be incorporated as a part of an autonomous robot navigation system. All the modules have been designed in synthesizable VHDL and the memory modules have been generated as Altera macro-blocks, everything developed in the Quartus II synthesis environment.

A. The log-polar transformation circuit

The base for transforming cartesian to log-polar images is a circuit for transforming cartesian to log-polar coordinates. The architecture for this image transformation circuit is shown in Fig. 2.

There are three blocks which form the circuit for image transformation: the counters, the LOG-CORDIC circuit and the memory. The counter block generates the cartesian coordinates at every clock cycle. At the same time, the corresponding I(x,y) grey level value enters the circuit. The (x,y) coordinates are transformed to log-polar coordinates at the LOG-CORDIC circuit; this circuit is discussed below. Finally, the grey level value is written to an internal memory at address (ξ,γ). This memory stores the resulting log-polar image and it is read out afterward for further processing.

The circuit for the memory is more complex than shown in Fig. 2, since the size of this memory is less than $\xi \times \gamma$ locations. The aim was not only to save memory bits in the circuit, but to solve a problem with the log-polar mapping. This problem has to do with the correspondence between one pixel in the cartesian plane to many of the log-polar plane; for example, the center pixel of the cartesian image plane, maps to the first ring in the center which has 128 pixels (considering there are 128

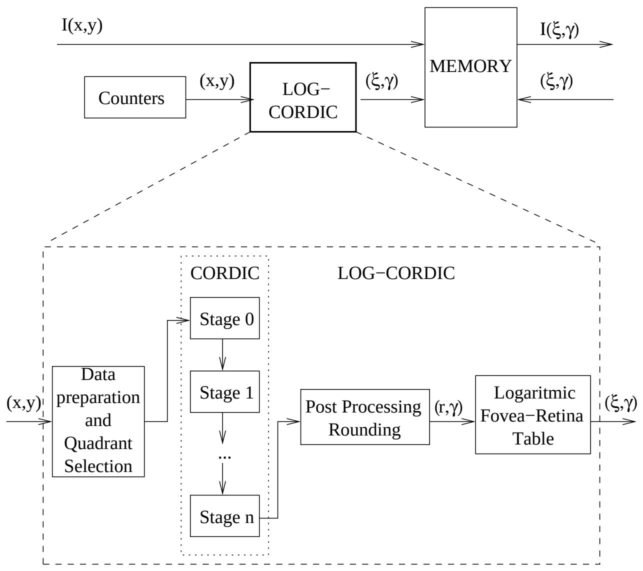

Fig. 2. Cartesian to log-polar transformation block diagram

pixels per ring). To solve this problem, pre-coders have been included for the memory addressing so there is only one byte addressed by a set of log-polar coordinates. In the example before, there is a single byte which is addressed by every of the 128 pixels of the first ring. Furthermore, one cartesian pixel can correspond to several log-polar pixels, which can be located at several rings, specially at the center where the log-polar pixels are usually smaller than cartesian pixels. The correspondence among these cartesian and log-polar pixels has been implemented with a previously calculated table.

The LOG-CORDIC circuit has the function of transforming cartesian to log-polar coordinates following the transformation (1). From this equation it is possible to see that the constants A and B are included in the ξ radial component. Furthermore, any amplification constant of the CORDIC circuit, as A_n, is also included in the ξ calculation. In the other hand, the only parameter affecting the γ angular component, is the number of pixels for each ring. All these factors must be included in the circuit.

The angular component is simply calculated using the CORDIC algorithm. The number of pixels per ring is coded in the circuit using the definition of the arctangent table. The CORDIC algorithm uses for the calculation of z_{i+1} a table of previously calculated values for all $\arctan 2^{-i}$. If this arctangent is calculated in the range of $[0..2\pi)$, the resulting angular parameter will be a number expressed in radians. But if tangents are calculated in the range $[0..\gamma_{max})$, where γ_{max} is the number of pixels per ring, the angular result will be directly the pixel in the ring without any further transformation. We then compute the table of arctangents taking into account the number of pixels per ring; if a different number of pixels per ring in the transformation is desired, it is necessary to reconfigure the circuit to accommodate the new arctangent table. All these changes to the angular parameter do not affect the radial component r which is used to calculate the log-radial component ξ.

The CORDIC algorithm directly gives the angular component for both polar and log-polar coordinates since $\theta = \gamma$, but it only gives the value of the radial component r of the polar coordinates, apart from some constant to be applied to this coordinate. The r coordinate must follow several transformations to get ξ. This transformation also needs a logarithmic operation. It is difficult to have a circuit that performs all these calculations, especially the logarithmic function.

Taking into account that there is a finite number of radial r and log-radial ξ components, it is simple to have a table to map r and ξ. This table has been previously calculated and it takes into account all the parameters and transformations between r and ξ. The complexity of this table is like 10% of a simple logarithmic circuit, and this last one would only calculate the logarithm and not all the other factors. Furthermore, any processing circuit would have a larger delay than this simple table.

The block diagram of the LOG-CORDIC circuit itself is also shown in Fig. 2. The first block calculates the quadrant where the cartesian point is located, and then moves the (x,y) coordinates to the first quadrant, since the CORDIC algorithm works at this quadrant. The second block is the CORDIC core itself working in vector mode; it is formed by several pipelined stages. Each stage calculates one of the steps of the CORDIC algorithm. Ten stages have been employed for this implementation. The third block puts the results in the correct quadrant and rounds them to the nearest 10-bits integer (the internal size for this CORDIC operator is 12 bits). The last block is the table that summarizes all parameters contributions and logarithmic-linear (retina-fovea) transformations.

This LOG-CORDIC circuit has been specified in VHDL and all the constant tables has been calculated and coded in a C++ program. The design takes its start point from an existing core of the CORDIC algorithm [11]. The data path for the input/output bus is ten bits wide, but the internal CORDIC data bus is 12 bits to allow good output results after processing. This number of bits is enough for transformations that are about 512×512 in the cartesian plane, and 76×128 in the log-polar plane. This number of bits is also good for larger cartesian images, up to 1024×1024, and 512×512 for log-polar, though it makes no sense to use a large log-polar size. With these sizes the complete LOG-CORDIC circuit occupies around 1.200 logic blocks of a device with more than 30.000 logic blocks. The CORDIC element itself occupies the larger part with 950 logic blocks. The circuit works at 60 MHz with no special refinement, since this speed is already around ten times above our requirements. Nevertheless, it is still possible to increase the CORDIC performance using an efficient mapping for the FPGA [12], [13].

B. Image processing stage

Both algorithms explained at section III have been implemented as the image processing stage of the system. In this way, the system can choose the most adequate processing algorithm.

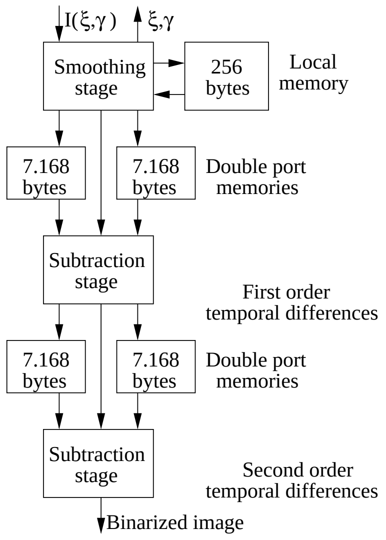

Fig. 3. Processing stage for the motion detection algorithm implementation

1) Motion detection algorithm synthesis: The algorithm explained in section III-A describes a differential algorithm for detecting objects that move respect the background, discarding automatically the image displacement due to the camera self movement when this movement is uniform along the camera optical axis. The algorithm has been divided into three stages that work simultaneously as a data pipeline as shown in Fig. 3. Double port memories of exactly the retinal image size (7.168 bytes) are placed between the stages in order to accelerate differential computations. Moreover, the first stage has 256 bytes of local memory for storing a log-polar ring. The ring values are shifted systolically with the aim of computing a grey level average value for accomplishing the smoothed image condition of the algorithm.

1) The first stage smoothes the original log-polar image storing the smoothed image in its double port memory and simultaneously giving it to the next stage. This smooth is made with an elementary nine pixels unitary convolution mask in order to simplify the computations.
2) The second stage computes the first temporal derivative as an image subtraction, accepting the smoothed pixel from the precedent stage. It also reads the double port memory where the precedent smoothed image is stored. Subsequently, first order differences are calculated as a simple pixel subtraction. Finally, the differential pixel is sent to the third stage, being simultaneously stored in its double port memory.
3) The last stage computes the second order temporal derivative and binarizes the image. This stage receives first order differences from the preceding one, simultaneously reads the differences previously stored at the double port memories. The second temporal derivative image is computed as differences between two first order temporal derivative images. Moreover, this value is compared to a threshold that is basically related to

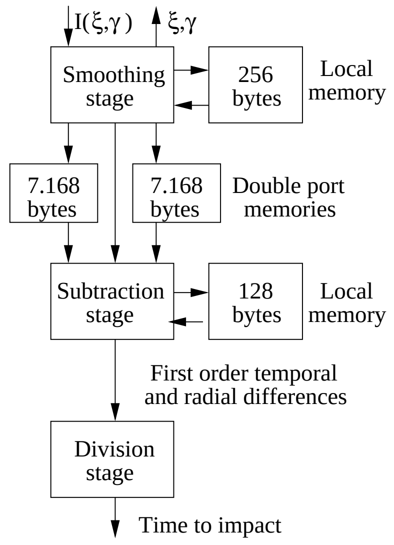

Fig. 4. Processing stage for the time to impact algorithm implementation

the camera movement and scene illumination. The final result is a sequence of binarized images which have marks for the points that belong to self-moving objects.

The algorithm has been successfully synthesized into the APEX20K device, occupying 230.400 RAM bits (70% of the total available RAM resources) and less than 1.000 LEs.

2) Time to impact computation algorithm synthesis: The algorithm for time to impact computation has been also implemented as image processing stage. It has been also implemented with the same methodology of splitting the overall task into a pipeline of stages, double port memories have been employed for accelerating the computation of the first temporal derivative. Again, the algorithm has been divided into three stages and there are two double port memories. Fig. 4 shows the algorithm implementation in the reconfigurable pipeline.

1) The first stage is exactly the same smoothing block designed for the previous algorithm.
2) The second stage computes the first temporal derivative and the radial gradient. The first order differentiation is computed through the same image subtraction policy described previously. Simultaneously, the radial gradient is computed with the smoothed pixel supplied by the previous stage, and the pixel corresponding to the inferior ring stored in a local small memory of 128 bytes.
3) Finally, the third stage computes the time to impact map for each pixel making an integer division of both values.

The algorithm has been also successfully synthesized into the APEX20K device, occupying near 115.000 RAM bits (35% of the total available RAM resources) and less than 1.100 LEs. So, it is also feasible to increase the algorithm accuracy improving the division stage.

V. Conclusions

Reconfigurable systems on-a-chip appear as a technology that combines hardware performance, software reconfigurability and low resources consumption. A log-polar image processing system has been developed in a SOPC. Space-variant vision has been employed to reduce the total amount of data to be processed, reducing the memory size for making possible the implementation of several local frame grabbers inside the chip.

A circuit for cartesian to log-polar image mapping has been presented. The core of this circuit is the LOG-CORDIC circuit which transforms cartesian to log-polar coordinates. We tailored the CORDIC algorithm for this special purpose and added extra circuitry for the logarithmic calculation taking into account all the complex factors of a log-polar transformation.

Furthermore, two different algorithms have also been implemented as processing stage in the reconfigurable SOPC, showing its flexibility. Both algorithms take advantage of image reduction and computation simplification. The synthesis results show that a complete image processing system can be synthesized in a high-density SOPC chip, including a complete PCI interface. Log-polar vision has been employed to reduce the total amount of data to be processed, decreasing the memory size for making it possible the implementation of several local frame grabbers inside the chip.

Acknowledgment

This work has been supported by the Spanish Ministerio de Ciencia y Tecnología project TIC2001-3546, EU FEDER and the Generalitat Valenciana project CTIDIA/2002/142.

References

[1] F. Ferrari, J. Nielsen, P. Questa and G. Sandini. *Space variant imaging*, Sensor Review, 15(2), 1995, 17-20.
[2] F. Pardo, B. Dierickx and D. Scheffer . *Space-Variant Non-Orthogonal Structure CMOS Image Sensor Design*, IEEE Journal of Solid State Circuits, 33(6), 1998, 842-849.
[3] J. Ruiz, C. Nowack and B. Schneider. *VIPOL: A Virtual Polar-Logarithmic Sensor*, In Scandinavian Conference on Image Analysis, SCIA'97, Finland, 1997, 739-744.
[4] J.E. Volder. *The CORDIC trigonometric computing technique*, IRE Trans. Elect. Comput. Vol. EC-8, 1959, 330-334.
[5] J.S. Walther. *A Unified Algorithm for Elementary Functions*, Proc. Joint Computer Conf. 1971, 379-385.
[6] Andraka, R.: A survey of CORDIC algorithms for FPGA based computers. In: International Symposium on FPGAs, Monterey, California, USA, ACM/SIGDA (1998)
[7] Antelo, E., Villalba, J., Bruguera, J., Zapata, E.: High Performance Rotation Architectures Based on Radix–4 CORDIC Algorithm. IEEE Transactions on Computers **46–8** (1997) 855–870
[8] W.G. Chen and N. Nandhakumar. *A simple scheme for motion boundary detection*, Pattern Recognition, 29(10), 1996, 1689-1701.
[9] J.A. Boluda and J. Domingo. *On the advantages of combining differential algorithms, pipelined architectures and log-polar vision for detection of self-motion from a mobile robot.*, Robotics and Autonomous Systems, Vol. 37(4), Elsevier, 2001, 283-296.
[10] M. Tistarelli and G. Sandini. *On the advantages of polar and log-polar mapping for direct estimation of time-to-impact from optical flow*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 15(4), 1993, 401-410.
[11] R. Herveille. *CORDIC Core Specification*. [Online]. Available at http://www.opencores.org/projects/cordic
[12] J. Valls, M. Kuhlmann and K. Parhi *A unified algorithm for elementary functions*, In IEEE Workshop on Signal Processing Systems (SIPS2000), Louisiana, USA, 2000, 336-345.
[13] T. Vladimirova and H. Tiggeler *FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm*, Military and Aerospace Programmable Logic Device, MAPLD'99, 1999.