A General Recursive Filtering Structure for Early Vision and its Hardware Architecture

R. DERICHE - H. GUIOT - G. RANDALL
INRIA - Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex - France

Abstract

We present the hardware architecture of an highly efficient recursive filtering structure. It allows to implement in a very efficient way several classical problems involved in early computer vision. Smoothing, differentiating and image edge extraction are done using this recursive filtering structure with a fixed number of operations per output element independently of the considered resolution. The differents parts composing the hardware architecture are identified and presented in detail.

1 Introduction

The importance of edge information has led to extensive research on their detection, description and use in computer vision systems. Several methods have been proposed and most of them suggest to first smooth the image, in order to remove noise, and second to consider estimates of the first or second derivative over some support as the appropriate quantity to characterize step edges. Respectively, peak and zero-crossings detections are then performed for the extraction step.

Recently, we proposed an highly efficient recursive filtering structure that allows to smooth, differentiates and extract edges in a very efficient way [1]. Here, we describe one possible hardware implementation of this filtering structure in order to be used as an edge detector stage working at a video rate. Such a performance is necessary for typical mobile robot applications.

A general outline of the edge extraction algorithm is given in figure 1. It consists of an initial convolution of the image with two separable filters in order to get the first directional derivatives. The gradient magnitude is then performed and points presenting a local maxima in the exact gradient direction are labeled as possible edges. An hysteresis thresholding, not considered in this architecture, is then applied to remove edges with low gradient magnitude.

The edge extraction is done using the optimal operator, with respect to localization and detection, that we have previously developed [3] by extending Canny's work [2] to Infinite Impulse Response filters. This filter presents good theoretical and experimental performance on noise suppression, detection and localization of edges in noisy signals. In the one-dimensional case, it corresponds to the first derivative operator and can be exactly implemented in a recursive way as follows :

$$y_1(n) = x(n-1) + 2e^{-\alpha} y_1(n-1) - e^{-2\alpha} y_1(n-2) \quad \text{for n} = 1,\dots,\text{M} \tag{1}$$

$$y_2(n) = x(n+1) + 2e^{-\alpha} y_2(n+1) - e^{-2\alpha} y_2(n+2) \quad \text{for n} = \text{M},\dots,1 \tag{2}$$

$$y(n) = (1-e^{-\alpha})^2 [y_2(n) - y_1(n)] \quad \text{for n} = 1,\dots,\text{M} \tag{3}$$

Relationships 1 through 3 give a very efficient procedure for calculating the first derivative of the input signal $x(n)$ at any resolution, fixed by the parameter α using only 5 multiplications and 5 additions per output element.

The following second order recursive implementation is used in order to implement the smoothing operator:

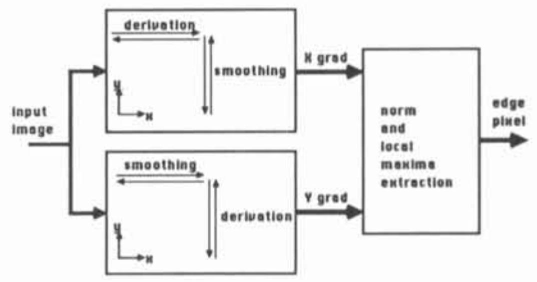

Figure 1: General overview of the edge extraction scheme.

$$y_1(n) = k[x(n) + e^{-\alpha}(\alpha-1)x(n-1)] + 2e^{-\alpha} y_1(n-1) - e^{-2\alpha} y_1(n-2) \quad \text{for n} = 1,\dots,\text{M} \tag{4}$$

$$y_2(n) = k[e^{-\alpha}(\alpha+1)x(n+1) - e^{-2\alpha}x(n+2)] + 2e^{-\alpha} y_2(n+1) - e^{-2\alpha} y_2(n+2) \quad \text{for n} = \text{M},\dots,1 \tag{5}$$

$$y(n) = y_1(n) + y_2(n) \quad \text{for n} = 1,\dots,\text{M} \tag{6}$$

$$k = \frac{(1-e^{-\alpha})^2}{1+2\alpha e^{-\alpha} - e^{-2\alpha}} \tag{7}$$

By adjusting the parameter α that determines the impulse response width, we can effectively control the size of our operator and thus the amount of noise suppression without increasing the number of operations per output. This shows the main advantage accrued in using the recursive implementation. Using a non-recursive implementation, a number directly proportional to the filter size would have been required to compute the convolution operation. As an example for $\alpha = .5$ a 16 bits direct convolution will require roughly 57 operations. For $\alpha = .25$, the computational effort increases to 113 operations per output element while it does not change for the recursive implementation.

To obtain the gradient in X, the image must be derived in the X direction and the result smoothed in the Y direction. To obtain the gradient in Y, the same process is repeated but with X and Y swapped. In order to derive the image in the X direction, each row is filtered from left to rigth using equation 1 and from rigth to left using equation 2. A temporary result is then obtained by subtracting both results as indicated by equation 3.

The smoothing filter is then applied to the columns of the temporary image obtained. Equation 4 is applied from top to bottom while equation 5 is applied from bottom to top. The final result is then obtained by application of equation 6.

All the filters used are second order recursive filters that can be implemented using the following general form:

$$y(n) = a_1 x(n) + a_2 x(n-1) + b_1 y(n-1) + b_2 y(n-2) \quad \text{for n} = 1,\dots,\text{M} \tag{8}$$

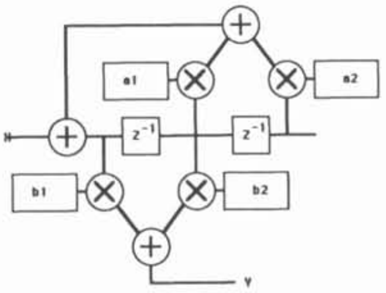

Figure 2: Canonical form of a second order recursive filter.

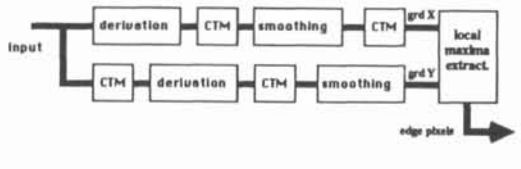

Figure 3: Video rate gradient computation.

It should be pointed out that for the filter given by 5, a delay has to be added in order to use this general structure. The coefficients used characterize the filter (smoothing or first derivative) and the parameter α determines the impulse response width of the operator. This structure is well adapted for the smoothing and derivativation operations and allows to deal with very large filters. This is not the case for a classical FIR implementation where the number of coefficients is fixed.

Figure 2 illustrates the canonical form of this general recursive filtering structure.

2 Hardware architecture

In order to reduce development cost and complexity we need to divide the system in some similar basic building blocks. Our first idea was to build blocks in order to do the basic functions of figure 1 (filtering and memory transposition). So we must have a basic second order filter (BSOF) and a memory that can take input data in row order and output them in column order, we name this last block: corner turner memory (CTM).

We can define each building block as a hardware module, one for filtering and one as CTM. Developing the hardware in this modular way seems easier and more flexible than trying to implement all the system in a big unit. If we use this approach, the edge detection at video rate can be done as shown in figure 3.

Figure 3 shows the implementation of the edge extraction module. The data flow is divided by two in order to parallelize the calculation of X and Y gradients. Derivation and smoothing are implemented with the same IIR block and appropriated coefficients. CTM blocks switch rows and columns addressing of data between input and output. The X and Y gradient data flow at the output of the filter block is directed to the *non maxima local extraction block* (NML), where pixels are labeled as possible edges before the hysteresis thresholding block.

Using the commutativity property of convolution operation leads to economize two CTM blocks as shown in figure 4. However, it should be pointed out that such a solution leads to work with the transposed image in the following modules. We must care to take into account this economical modification.

Another implementation is illustrated in figure 5. The data flow concept is not preserved here. The input image is stored in the *multiple frame memory* (MFM). Derivative and smoothing filters in X and Y directions are sequentially applied to the data. Intermediate results are stored

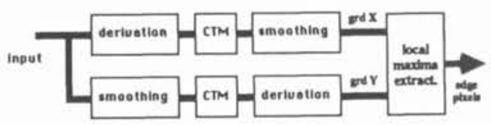

Figure 4: Minimal video rate gradient computation.

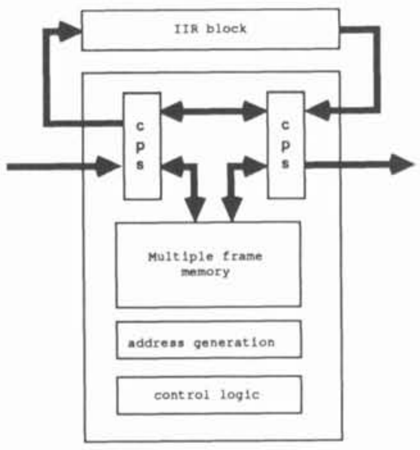

Figure 5: gradient extraction with two boards.

in the MFM. The IIR block is the same as in the previous approach. The data move between filter and memory boards until total processing of the image. The cross port switch changes the data pathway and the address generator switches rows and columns when necessary. This approach is cheaper (it uses only one IIR and memory blocks) but does not work at video rate.

2.1 Corner turning memory

The IIR filter works sequentially with flow of data. The corner turning memory (CTM) exchanges rows and columns data in order to transpose the image and to access the columns in a sequential way.

Figure 6 shows the principle of the CTM block. For each clock cycle, one pixel of image at time n is read and output and one pixel of image at time n+1 is input and written at the same address. Every new frame cycle, the address generator switchs from present addressing mode to orthogonal addressing mode. This solution introduces one frame delay in the edge extraction pipeline at this step of the computation.

2.2 IIR block architecture

High speed monolithic IIR building blocks are not commercially available at this date. One possible solution is to use FIR commercial available blocks (IMS A100, ZORAN, LSI FIR chip, etc.) in order to realize an IIR filter or to approximate it. But there are two major problems: speed, and coefficients and data width. It is possible to implement an IIR filter by using two FIR building blocks or by swapping the coefficient register bank in a FIR building block of type IMS A100. Another approach is to implement the algorithm with an FIR filter but all the generality of the recursive implementation is lost. FIR modules operating at video rate are available, but only with 4 to 8 coefficients on 12 bits, this word width is too small for our stereo vision application.

Our proposed IIR block is the most general form of the filter, it allows a great variation of the resolution parameter

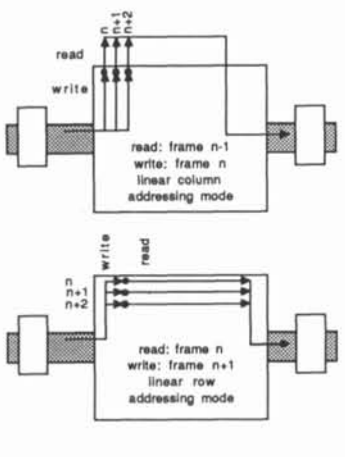

Figure 6: The corner turning memory process.

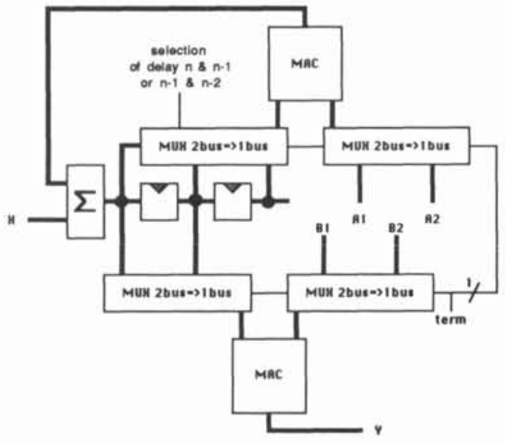

Figure 7: Hardware implementation of a second order IIR filter

o without loss of precision in the position of the edges and without instability.

Figure 7 illustrates the hardware implementation proposed. New input data is clocked into the filter every 150 ns. The filter cell works twice faster in order to multiply the delayed data by appropriated filter coefficients and adds it in an MAC's accumulator. In order to implement the filter equations, the appropriated filter coefficients and the number of delay taps are programmed.

Two multiplier-accumulator building blocks (MAC) working at 35ns, are used. This performance and the use of multiplexers permits to economize two MACs. A sequencer controls the filter cell. It has three functions:

- Determination of te right number of delay taps in X data pathway according to the filter used.
- Swapping of the MACs inputs in order to multiply delayed data by appropriated coefficients.
- Inhibition of MAC accumulation every even points.

Figure 8 illustrates the architecture of the IIR general module. The structure is duplicated in order to work in a flip-flop way.

Two IIR cell filters are present and work with successive rows of the image (or columns if a CTM has made the necessary inversion). A temporary row memory permits the filtering in both directions and the output adder is used to compute the results.

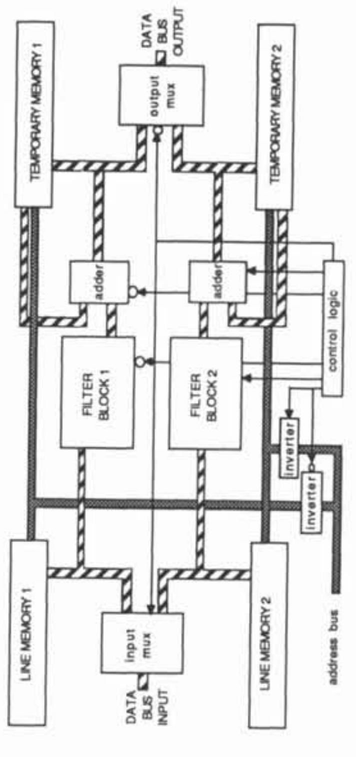

Figure 8: IIR board architecture

- Data line n is input through the input multiplexer and directed to the line memory 1 and through the filter block 1 to the temporary memory 1. The adder is inhibited and the output multiplexer is connected to the other side of the module.

  Data is write in line memory 1 and temporary memory in the same direction and the filter order is selected according to the application.
- The address bus is complemented. The filter order and coefficients are changed. Data stored in memory line 1 is filtered and added with temporary memory 1 content. The result of the addition is directed to the output multiplexer.

The other side of the block works identically with line n-1.

2.3 Local maxima extraction

The non maxima suppression scheme uses a nine-pixel neighborhood and requires three points, one of which will be the current point, and the other two should be estimates of the gradient magnitude at points displaced from the current point by the vector normal to the edge direction.

If the gradient of the current point is greater than its interpolated neighbors, the current point is then labeled as a possible edge point. The difficulty of this algorithm is that it computes an interpolation operation. This problem is solved by the hardware structure of figure 9. At the input of this stage, two lookup tables are present.

- A gradient module LUT. It gives the norm of the gradient magnitude to the neighborhood structure.
- A gradient direction and sense LUT which controls an 8 to 4 multiplexer in order to send to the arithmetic unit the neighbors to be interpolated. Both interpolations are made in parallel. The output comparison is made with the module norm of the current point. The output of this block is boolean and differentiates edge points and no edge points.

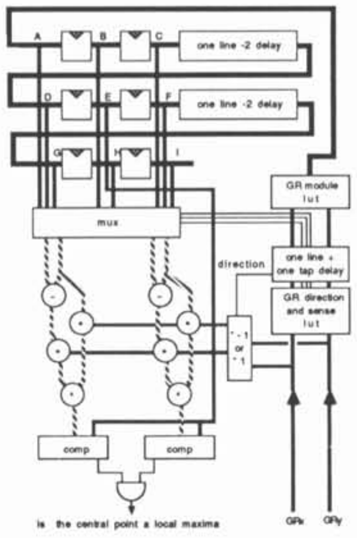
A
B
C
one line -2 delay
D
E
F
one line -2 delay
G
H
I
GR module lut
mux
direction
one line + one tap delay
GR direction and sense lut
-1 or +1
comp
comp
is the central point a local maxima
GRx
GRy

Figure 9: Local maxima extraction

frame delay
deriv1
CTM1
smooth1
smooth2
CTM2
deriv2
local maxima extraction
one frame delay in local maxima extraction
for lines delay in the filter before begin local maxima extraction
one frame delay in the CTM
one line delay in the filter
two frames plus five lines

Figure 10: Timming diagram

In figure 10 we give a global timing description of the gradient extraction. Because of the necessity of an inversion between row and column at every pass of the filter, we must wait for the completion of the first filtering pass before beginning the second pass. So we can see that this pipeline introduces 2 images and 5 lines delay between input and output, even if we compute everything at video rate. In figure 10 we can read in Y axis the board name, in X axis the time and the solid line represents the occupation time of each module for current image.

3 Simulation results

We have tested the smoothing and derivative filters using a fixed-point implementation for various number of bits and values of the parameter α. The results obtained are very similar to those of a floating point implementation when using values for α close to one for a number of bits bigger than 10. Dealing with values of α close to 0.5 and less, provides a need to increase the number of bits to 16. The normalized mean square error (NMSE) between the floating point impulse response of the derivative filter and its approximated version using a fixed point implementation with 16 bits is shown in figure 11, for different values of α varying from .1 to 1.4. This figure establishes the limits

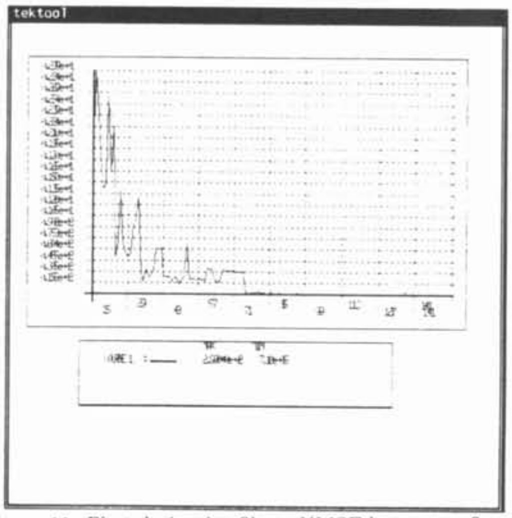

Figure 11: First derivative filter: NMSE between a floating point and a 16 bits fixed point implementation for values of α varying from .1 to 1.4

over which it is valid to use α. If α is selected less than 0.2 the impulse response used will deviate from those of the floating point implementation.

4 Conclusion

In this paper, we examined a possible hardware implementation of a general recursive filtering structure. It allows to implement in a very efficient way several classical problems involved in early computer vision. Its application to the problem of edge detection has also been considered.

References

[1] R.Deriche. Fast Algorithms For Low-Level Vision. *Proceedings 9th International Conference on Pattern Recognition*, 14-17 Nov 1988, Rome.

[2] J.F.Canny. *Finding Edges and Lines in Images.* Technical Report 720, MIT, Artificial Intelligence Laboratory, Cambridge, Massachussets, June 1983

[3] R.Deriche. Using Canny's criteria to derive a recursively implemented optimal edge detector. *The International Journal of Computer Vision*, 1(2), May 1987.