

# Real-time 2-D Feature Detection on a Reconfigurable Computer

A. Benedetti<sup>†</sup>

P. Perona<sup>†‡</sup>

<sup>†</sup>California Institute of Technology, Pasadena, CA 91125

<sup>‡</sup>Università degli Studi di Padova, Italy  
{arrigo,perona}@vision.caltech.edu

## Abstract

**We have designed and implemented a system for real-time detection of 2-D features on a reconfigurable computer based on Field Programmable Gate Arrays (FPGA's). We envision this device as the front-end of a system able to track image features in real-time control applications like autonomous vehicle navigation. The algorithm employed to select good features is inspired by Tomasi and Kanade's method. Compared to the original method, the algorithm that we have devised does not require any floating point or transcendental operations, and can be implemented either in hardware or in software. Moreover, it maps efficiently into a highly pipelined architecture, well suited to implementation in FPGA technology. We have implemented the algorithm on a low-cost reconfigurable computer and have observed reliable operation on an image stream generated by a standard NTSC video camera at 30 Hz.**

## 1 Introduction

Feature selection and tracking is a fundamental problem in computer vision research. By measuring the displacement of features in the sequence of images seen by a camera, it is possible to recover information both on the structure of the environment and on the motion of the viewer. Thus, a feature tracker is the front-end for any system employed to solve many practical control problems, like autonomous navigation and "structure from motion". In this paper, we pay attention to feature selection because it is the most computationally intensive task to be carried out by a feature tracking system; we believe that once the locations of the features are known at each frame, a conventional processor will be able to solve the correspondence problem. To the best of our knowledge, the only real-time feature selection system described in literature is the system employed by the ASSET-2 [1] motion segmentation and shape tracking system. This system comprises four VME boards, equipped with custom hardware circuits performing the different stages of the Harris corner finding algorithm [2]. The system that

we propose here, on the other hand, has been implemented on a low-cost reconfigurable computer based on Field Programmable Gate Arrays (FPGA's). Reconfigurable computers are sufficiently flexible to permit the implementation of new algorithms on existing hardware, and their performance is adequate for real-time operation of many image processing problems [3]. Our feature detection system is based on the work of Tomasi and Kanade [4]. We have decreased the complexity of this method by eliminating the need for any transcendental arithmetic operations. This simplification has led to the implementation of the algorithm on six Xilinx FPGA's [5] hosted by a Gigaops Spectrum G800 PC board [6]. The rest of the paper is organized as follows. In the next section the feature selection method is presented in detail. In Sec. 3 we give a brief overview on FPGA-based custom computing machines, while in Sec. 4 we present the details of the implementation of the feature selection system. Finally, in Sec. 5 experimental results are reported and some conclusions are drawn.

## 2 Description of the method

The feature selection method that we have implemented is based on the work of Tomasi and Kanade [4], which we will briefly review here. Let  $I(x, y, t)$  denote the continuous function defining the brightness values of a sequence of images seen by a camera. The partial derivatives of  $I(x, y, t)$  with respect to  $x, y, t$  are denoted respectively by  $I_x, I_y$  and  $I_t$ . The idea underlying this method is that a good feature is a feature that is easy to track across several frames. Due to the presence of noise and distortion it is not practical to track individual pixels, and for this reason patches of pixels are actually considered. Under the assumption that the brightness of the pixels forming the patch is constant, we can write

$$\frac{dI}{dt} = \nabla I^T \mathbf{v} + I_t = \mathbf{0}, \quad (1)$$










 a) $\lambda_1 = 16.9 \cdot 10^3$ $\lambda_2 = 78.3 \cdot 10^3$	 b) $\lambda_1 = 19.2 \cdot 10^3$ $\lambda_2 = 60.9 \cdot 10^3$	 c) $\lambda_1 = 29.6 \cdot 10^3$ $\lambda_2 = 95.5 \cdot 10^3$
 d) $\lambda_1 = 1.07 \cdot 10^3$ $\lambda_2 = 65.4 \cdot 10^3$	 e) $\lambda_1 = 528.5$ $\lambda_2 = 45.7 \cdot 10^3$	 f) $\lambda_1 = 1.5 \cdot 10^3$ $\lambda_2 = 14.8 \cdot 10^3$
 g) $\lambda_1 = 309.9$ $\lambda_2 = 1759.2$	 h) $\lambda_1 = \mathbf{1069.1}$ $\lambda_2 = 3808.8$	 i) $\lambda_1 = 1401.6$ $\lambda_2 = 2515.4$

Figure 1: Sample patches and corresponding eigenvalues of  $G$

where  $\mathbf{v} = (dx/dt, dy/dt)$  is the velocity of every pixel belonging to the window that we are tracking. In other words, Eq. (1) expresses the conservation of the optical flow for points of patches with constant brightness. If we suppose that all the  $N$  points in the patch are moving at the same velocity, which is reasonable for small inter-frame displacements, we can formulate the linear problem

$$\begin{bmatrix} I_x^1 & I_y^1 \\ \vdots & \vdots \\ I_x^N & I_y^N \end{bmatrix} \mathbf{v} = - \begin{bmatrix} I_t^1 \\ \vdots \\ I_t^N \end{bmatrix},$$

that is expressed in matrix notation as

$$\mathbf{A}\mathbf{v} = \mathbf{b}. \quad (2)$$

The displacement  $\mathbf{v}$  is given by the linear least squares solution of Eq. (2), i.e.

$$\mathbf{G}\mathbf{v} = \mathbf{A}^T \mathbf{b}, \quad (3)$$

where

$$\mathbf{G} = \mathbf{A}^T \mathbf{A} = \begin{bmatrix} \sum_{k=1}^N (I_x^k)^2 & \sum_{k=1}^N I_x^k I_y^k \\ \sum_{k=1}^N I_x^k I_y^k & \sum_{k=1}^N (I_y^k)^2 \end{bmatrix} = \begin{bmatrix} a & \mathbf{1} \\ b & \mathbf{1} \end{bmatrix} \quad (4)$$

The matrix  $G$  and the linear transformation  $\gamma: V \rightarrow V$  associated with it tell us a great deal about the structure of the pixel neighborhood. Being symmetric, its eigenvalues  $\lambda_1, \lambda_2$  are equal to its singular values, that are related to the dimension of the range of  $\gamma$ . For the sake of clarity, we first consider the case where we are interested in the zero/non-zero pattern of the two eigenvalues. When one eigenvalue is zero and the other is greater than zero, for instance,  $G$  has rank one, thus there is a subspace of  $V$  of dimension one that cannot be reached when solving Eq. (3). This situation is common with one-dimensional features like edges in synthetic images, where the unreachable subspace corresponds to the component of the displacement  $\mathbf{v}$  parallel to the direction of the edge (aperture problem). When dealing with real images we have to take into account the presence of noise and the fact that there might be cases where  $G$  is almost rank deficient, yet its singular values are nonzero. When the smallest eigenvalue is close to zero, for instance,  $G$  is close to the set of  $2 \times 2$  matrices with rank one, i.e.

$$\min_{\text{rank}(\mathbf{B})=1} \|\mathbf{G} - \mathbf{B}\|_2 = \lambda_1 \simeq 0. \quad (5)$$

(For a proof, see [7, Sec. 2.5.5]). If the other eigenvalue is several orders of magnitude greater it is readily seen that  $G$  is distant from the null matrix  $0$ , in fact

$$\|\mathbf{G} - 0\|_2 = \|\mathbf{G}\|_2 = \lambda_2 \gg 0. \quad (6)$$

Therefore, the conditions  $\lambda_2 \gg \lambda_1 \simeq 0$  and  $\text{rank}(G) = 1$  are both related to patches with a unidirectional texture, like those in Fig. 1-d,e,f. Reasoning along these lines, we see that two small eigenvalues correspond to patches with approximately constant brightness, like those depicted in Fig. 1-g,h,i, while two large eigenvalues correspond to salt-and-pepper textures, corners or other patterns where the brightness gradient has a strong signal across two nearly orthogonal directions across the neighborhood (see Fig. 1-a,b,c). In principle, a way to select good features is by requiring  $G$  to be well conditioned, i.e. enforce an upper bound on the condition number:

$$\text{cond}(\mathbf{G}) = \frac{\lambda_2}{\lambda_1} < c_M. \quad (7)$$

Although effective with synthetic images, the strategy expressed by (7) does not give good results with real images. Due to the presence of noise, in fact, the condition number associated with the patches depicted

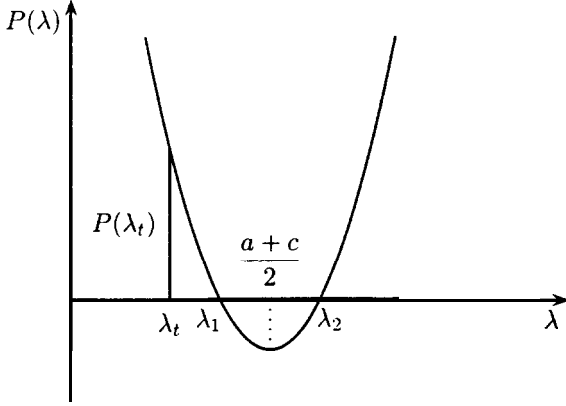


Figure 2: Requiring  $\lambda_1 > \lambda_t$  is equivalent to the conditions  $P(\lambda_t) > 0$  and  $\frac{a+c}{2} > \lambda_t$ , as shown in this picture.

in Fig. 1-g,h,i is low, even if their brightness profile is constant. For this reason, Tomasi and Kanade have proposed the following criterion:

$$\min \{\lambda_1, \lambda_2\} = \lambda_1 > \lambda_t. \quad (8)$$

By imposing a lower bound on  $\lambda_1$  it is ensured that

1.  $\lambda_2 > \lambda_1 > \lambda_t$ , i.e. G is distant from the set of singular matrices. In other words, the variations in the brightness profile are above the image noise level.
2. G is also distant from any rank deficient  $2 \times 2$  matrix and, as long as  $\lambda_2 \simeq \lambda_1$ , it represents a linear map that is not stretched in any specific direction. Note that  $\lambda_2$  is always bounded by a function of the maximum brightness value.

The method proposed in [4] for selecting 2-D features can be summarized as follows:

1. Compute the image gradient  $\nabla I(i, j) = (I_x(i, j), I_y(i, j))$  across the image at every pixel location  $(i, j)$ ;
2. Calculate  $a(i, j), b(i, j), c(i, j)$  as defined in Eq. (4) for every  $N$  pixels mask centered at the current pixel  $(i, j)$ ;
3. Find  $\lambda_1(i, j)$ , i.e. associate every image pixels with the minimum eigenvalue defined by

$$\lambda_1 = \frac{a+c}{2} - \sqrt{\left(\frac{a+c}{2}\right)^2 - (bc - b^2)}; \quad (9)$$

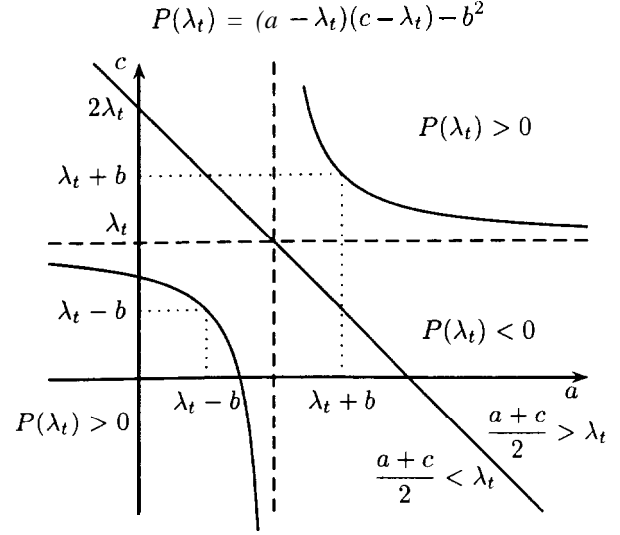


Figure 3: The condition  $\frac{a+c}{2} > \lambda_t$  is equivalent to either  $a > \lambda_t$  or  $c > \lambda_t$ , as we only need to check if the point with coordinates  $(a, c)$  is in the region of the plane defined by  $P(\lambda_t) > 0$ .

4. Discard any pixel with

$$\lambda_1(i, j) < \lambda_t;$$

5. Perform a final non-maximum suppression step on  $\lambda_1(i, j)$ , yielding the actual feature locations.

When we consider the real-time implementation of this algorithm, the most critical step is the calculation of  $\lambda_1$ , as defined by Eq. (9). Due to its computational complexity, in fact, the square root operation is hard to implement on FPGA's [8]. For this reason we have devised a method, described in the next section, that does not require its calculation, yet gives the same features found by Tomasi and Kanade's algorithm.

### 2.1 Decreasing the complexity

If we call  $\lambda_1$  the smallest root of the characteristic polynomial

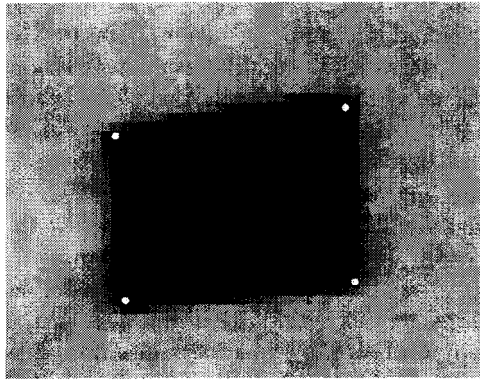
$$P_\lambda = (a - \lambda)(c - \lambda) - b^2, \quad (10)$$

we can see that the condition expressed by Eq. (8) is equivalent to

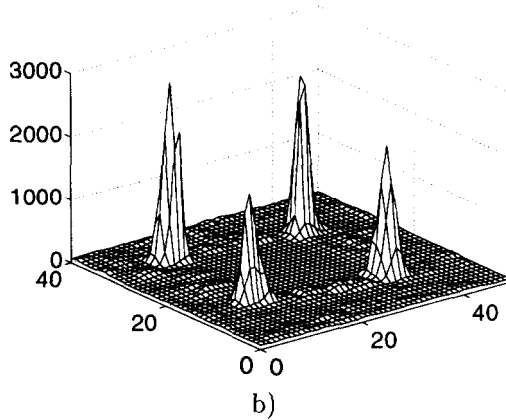
$$P_{\lambda_t} > 0 \text{ and } \frac{a+c}{2} > \lambda_t, \quad (11)$$

as shown in Fig. 2. Moreover, we can further simplify (11) by observing that  $\frac{a+c}{2} > \lambda_t$  merely separates the regions defined by  $P_{\lambda_t} > 0$  in the  $(a, c)$  plane, and is thus equivalent to either

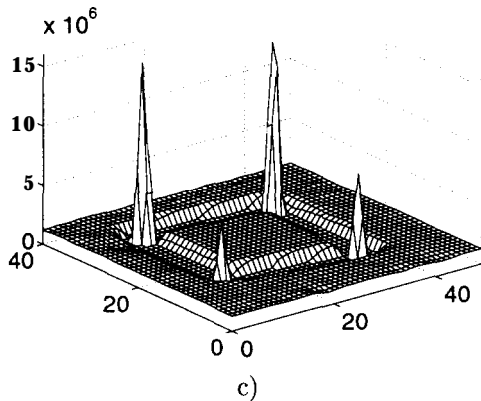
$$a > \lambda_t \quad (12)$$



a)



b)



c)

**Figure 4: In b) and c) we show respectively the  $\lambda_1(i, j)$  and  $P_{\lambda_t}(i, j)$  (see Eqs. (9) and (14),  $\lambda_t = 10^3$ ) surface plots computed on pixel data from image a). It is manifest that  $P_{\lambda_t}(i, j)$  has a strong signal at the four corner locations shown in a).**

or

$$c > \lambda_t.$$

In the following, we will use the former inequality. Thus a pixel is a feature candidate if

$$P_{\lambda_t}(i, j) > 0 \text{ and } a(i, j) > \lambda_t. \quad (13)$$

The reduction in complexity achieved when comput-

ing  $P_{\lambda_t}(i, j)$  instead of  $\lambda_1(i, j)$  is considerable, especially when we consider a direct hardware mapping of the method. Comparing Eq. (9) with Eq. (13), we see that the former requires three multipliers and a circuit dedicated to the calculation of the square root, while the latter only needs two multipliers. When we avoid computing  $\lambda_1(i, j)$ , steps 4 and 5 in the algorithm outlined above have to be formulated in a different way. We have observed on a variety of images that qualitatively similar results can be achieved by thresholding  $P_{\lambda_t}(i, j)$  and performing the non-maximum suppression step on the same function. The steps of the simplified algorithms are the following:

1. Compute  $\nabla I(i, j) = (I_x(i, j), I_y(i, j))$ ;
2. Calculate  $a(i, j), b(i, j), c(i, j)$ ;
3. Find

$$P_{\lambda_t}(i, j) = (a - \lambda_t)(c - \lambda_t) - b^2; \quad (14)$$

4. Retain pixel  $(i, j)$  iff

$$P_{\lambda_t}(i, j) > 0 \text{ and } a(i, j) > \lambda_t; \quad (15)$$

5. Discard any pixel that is not a local maximum of  $P_{\lambda_t}(i, j)$ .

Even though the test expressed by Eq. (15) is quantitatively different from Eq. (8), we have observed a very strong similarity between the two quality indicators, as shown in Fig. 4. In Fig. 5 and 6, the features extracted by the simplified algorithm from two real images taken by a camera are shown.

### 3 FPGA-based custom computers

A system designer is constantly faced with a tradeoff between performance and generality: a digital system designed to handle different types of tasks is usually slower than a system tailored to a specific application. This is the reason why computationally intensive tasks are handled by dedicated system architectures, usually implemented in Application Specific Integrated Circuits (ASIC's). Although they offer more than adequate performance, ASIC's often lack the ability to adapt themselves to a changing environment. Moreover, ASIC design has high non-recurring costs and requires large engineering effort. A new methodology for the implementation of digital logic circuits, based on Field Programmable Gate Arrays (FPGA's), emerged during the mid 80's. The basic architecture of an FPGA consists of a large number of Configurable Logic Blocks (CLB's) and a programmable mesh of interconnections. Both the function performed by the logic blocks and the interconnection pattern can be

specified by the circuit designer. In the beginning, FPGA's were mostly viewed as large Programmable Logic Devices (PLD's), thus they were usually employed for the implementation of the so-called "glue-logic" used to tie together complex VLSI chips like microprocessors and memories in general purpose computer systems. While several FPGA's were configured by static RAM (SRAM) cells, this was generally considered a limitation by users concerned about the chip's volatility. For this reason, fuse-based FPGA's were also developed, and for many applications were much more attractive because there was no volatility in their configuration. In the late 80's and early 90's it became clear that the volatility of SRAM-based FPGA's was not a liability, but could open an entirely new spectrum of applications. Since the programming of such FPGA's could be changed electrically at almost any time during operation, they found increasing use as a medium for the rapid prototyping of complex systems. Real-time image processing has been one of the first fields to exploit the advantages offered by this technology [9]. The amount of logic available in a single FPGA device has been increasing steadily for the last 10 years. Nevertheless, until last year it was unlikely that a system performing a nontrivial task would fit into a single device. For this reason there has been considerable interest in the so-called multi-FPGA systems, i.e. systems where the amount of logic borne by a single FPGA is effectively augmented by other devices. In multi-FPGA systems data flow through dedicated interconnection resources, and dedicated I/O subsystems are able to feed the information to be processed at high data rates. In the next section we give a brief description of the system that we have chosen for the implementation of the feature selection algorithm.

## 4 Description of the system

We have implemented the feature selection method described in Sec. 2 on the Gigaops Spectrum G800 multi-FPGA system [6]. The G800 system is a PC expansion board containing two FPGA's handling video and host communication, some external interface circuitry and four locations where expansion modules called XMOD's can be plugged in (see Fig. 7). XMOD's are cards that can be added to the system to add logic and memory resources. Up to four modules can be stacked in each location, thus a total of 16 add-on boards can be combined in a single system. These logic boards are connected together, and to the FPGA's on the base board, by a set of global buses. At one end of the G800 board it is possible to plug a video decoder/encoder interface board called VIDMOD, which can translate a standard NTSC video signal into the 4:2:2  $YCbCr$  digital video format. This makes it possible to use

this system for many different video processing applications, including computer vision. The topology of this architecture is a hybrid between a crossbar and a mesh: the two FPGA's on each XMOD (called respectively XFPGA and YFPGA) are directly connected in a mesh topology, while all the XFPGA's are connected to a 64 bit bus, called XBUS. The VMC FPGA, which is on the main board, is in turn connected to this bus and provides both routing and logic resources. This FPGA is also connected to the video decoder and encoder chips on the VIDMOD interface board, and is employed to transmit real-time video data onto the YBUS. This 32 bit bus is connected to every YFPGA in the system, making it possible to perform different operations on the video data stream in parallel. The HBUS is dedicated to communication with the host through the VL FPGA and FPGA configuration and monitoring.

### 4.1 System design issues

The complexity of the system dictated the use of modern logic synthesis tools throughout all design phases. Synopsys' FPGA Compiler [10] has been used for most part of the logic design starting from descriptions specified in the VHDL hardware description language.

Design of FPGA-based computing machines is carried out through the following steps:

1. Definition of data processing resources;
2. Determination of the precision of Intermediate results;
3. Definition of memory resources;
4. Design of the control logic.

The result of the first step depends on a tradeoff between the complexity of the algorithm and the available logic and communication resources, e.g. the number of FPGA devices, their capacity, and the number of interconnection lines between programmable devices. The FPGA's currently employed in our system-according to the manufacturer-contain the equivalent of  $20/28 \times 10^3$  logic gates. Moreover, the density of current FPGA's is increasing steadily, and FPGA manufacturers are announcing the availability of a new families of devices integrating up to 1 million logic gates. For this reason it is clear that the fundamental limitation of current and future reconfigurable systems is not in the total amount of logic borne by each FPGA, but in the constraints imposed by the interconnection topology. From our experience the scarce availability of interconnection resources in the G800 system forces the designer to duplicate logic functions in several FPGA's, thus leading to decreased FPGA utilization.

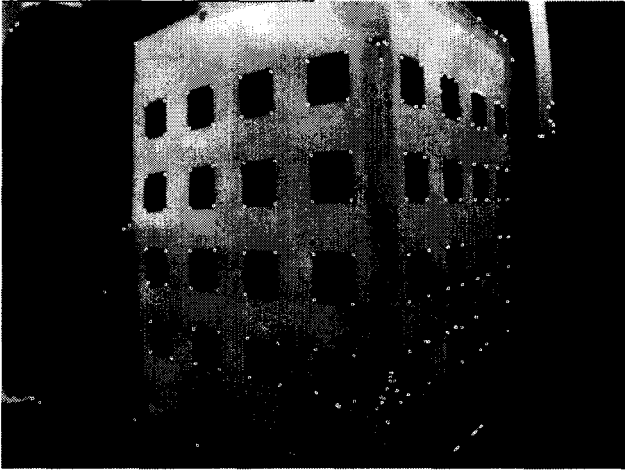


Figure 5: Features extracted according to Eq. (15) on a real image.

The determination of the bit widths of intermediate operands is extremely important, due to the fact that the logic density of FPGA devices is low when compared to an ASIC fabricated in the same manufacturing technology. We have observed, for instance, that the operands in the datapaths computing  $a$ ,  $b$  and  $c$  do not exploit their full dynamic range. This happens because they reach their maximum magnitude only for “salt-and-pepper” textures, i.e. textures with pixels taking values at the opposite ends of the allowed range. The following is an example of such a feature:



These types of features are very unlikely to be found in real images: moreover, a feature like the one depicted above would be very difficult to track, because even a small misalignment between two image frames would result in a very different intensity pattern. If we consider the term  $a$  as defined by Eq. (4) we see that 19 bits (including the sign bit) are needed to represent its value, since

$$a = \sum_{k=1}^9 (Y_W - Y_B)^2 = 9 \cdot (235 - 16) = 431649,$$

where  $Y_W$  and  $Y_B$  are respectively the luminance values for White and Black as defined by the 4:2:2  $YC_bC_r$  digital video standard. From the analysis of several real images digitized from a video camera we have seen that 17 bits are more than adequate for representing this value. If an overflow condition is detected, we simply discard the corresponding feature. This reduction in operand size leads to considerable savings,



Figure 6: Features selected on an image taken from a moving vehicle.

especially in the number of configurable logic blocks needed to synthesize the delay lines used to form the  $3 \times 3$  neighborhood for the computation of  $a$ ,  $b$  or  $c$  as defined by Eq. (9). The data processing resources required by this application are the adders, comparator and multipliers needed to process the incoming pixel stream according to Eqs. (4,10,12). Adders are automatically inferred by the synthesis tool from VHDL code and mapped to elements of the Xilinx X-BLOX library. This library contains adders of different bit widths exploiting the dedicated carry logic chain of the XC4000 family. The multipliers generated by the synthesis tools, on the other hand, are not optimized for this FPGA architecture. Therefore, for the calculation of  $(I_x)^2$ ,  $I_x \times I_y$ ,  $(I_y)^2$  and  $P_{\lambda_t}$  we have employed the parameterizable parallel multipliers generated by the Xilinx DSP CORE Generator, that were instantiated as VHDL components. These multipliers, tuned to achieve maximum performance on the Xilinx architecture, were the key elements for real-time operation of the system. The basic memory resource required by any FPGA based image processing system operating on pixel neighborhoods is the pixel line delay, that is usually implemented either with external RAM memory devices or with FPGA internal memory. The Xilinx XC4000 architecture, in fact, allows to configure every CLB as a  $37 \times 1$  bit shift register, and several registers can be cascaded in order to build FIFO memories of various widths and depths. The main advantage of this approach over external memory chips is the ability to tailor the width of the FIFO memory to the size of the operands, whereas the width of the data words that can be stored in an external memory chip is fixed.

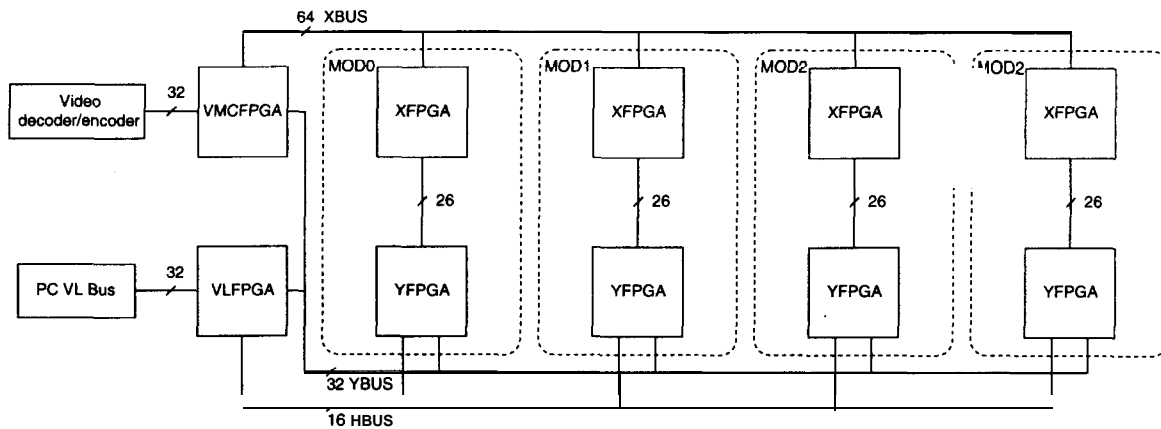


Figure 7: Schematic block diagram of the Gigaops G800 Reconfigurable Computer.

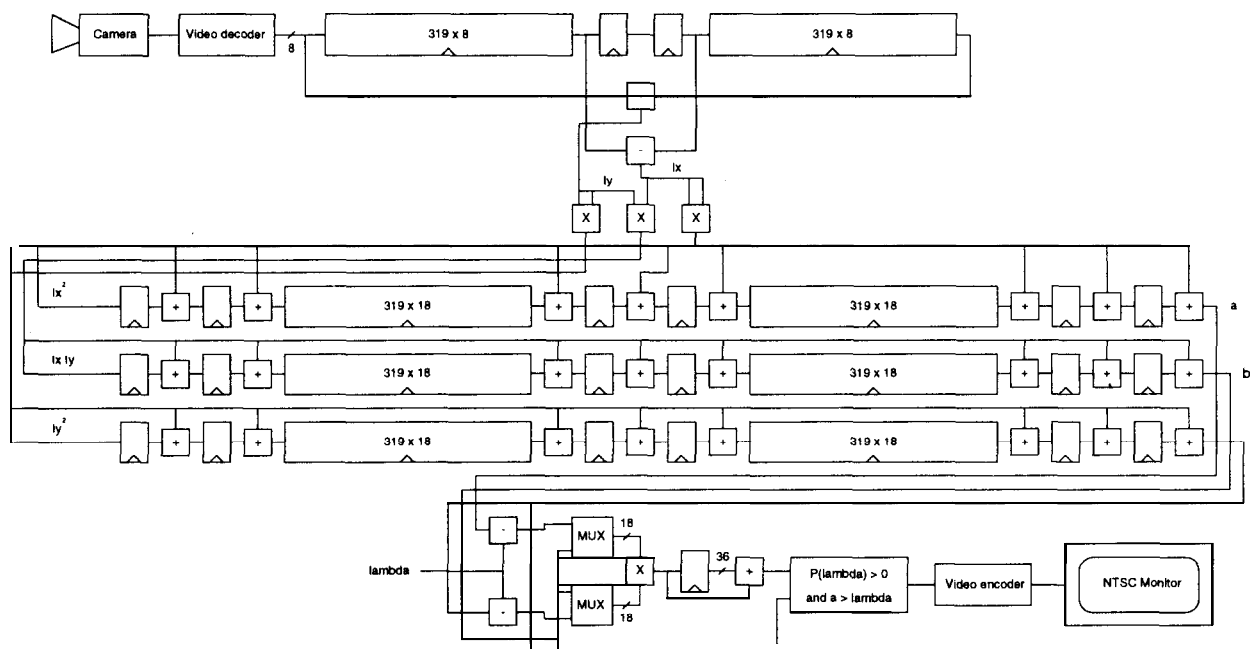


Figure 8: Schematic logic diagram of the real-time feature selection system.

#### 4.2 Implementation details

The data flow of many image processing systems can be decomposed as a sequence of operations on sets of data whose organization resembles that of the initial image. The first stage of the feature selection system, for example, computes the image gradient components operating on the pixel stream transmitted in raster scan order by the video decoder.  $I_x$  and  $I_y$  are computed convolving the input image respectively with

the kernels  $\begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix}$  and  $\begin{bmatrix} 0 & 0.5 & 0 \\ 0.5 & 0 & 0.5 \end{bmatrix}$  Two long

shift registers are used to delay the incoming pixel stream in order to make available at every clock cycle the intensity values  $I(i, j-1)$ ,  $I(i, j+1)$ ,  $I(i-1, j)$  and  $I(i+1, j)$ , where  $(i, j)$  are the coordinates of the central pixel of the two convolution masks. The main advantage of this scheme is that after a latency of two scan lines, needed to fill the two  $319 \times 8$  bit delay lines shown in Fig. 8, the  $I_x(i, j)$  and  $I_y(i, j)$  streams are synchronized, thus they can feed the next processing stage. The calculation of  $a$ ,  $b$  and  $c$  is performed in parallel by three chains of adders interleaved with pixel and line delay elements in order to build a  $3 \times 3$

mask in the  $(I_x)^2, I_x \times I_y$  and  $(I_y)^2$  planes. All delay lines were implemented using a feature of Xilinx FPGA's that allows the configuration of the internal logic function generators as a 32 x 1 bit static RAM cell. Four XC4028EX and two XC4020E Xilinx devices were employed for this application. In order to fit the 18 bit wide delay lines needed for computing  $a, b$  or  $c$  in a single FPGA device, we had to down-sample the NTSC 640 x 480 image by a factor of 2 along the  $x$  axis. Due to interlacing, the video decoder sends each field one at a time, so the system is actually processing 320 x 240 pixel images. The rest of the system presented in Fig. 8 calculates the value of  $P_{\lambda_t}(i, j)$  by time-multiplexing an 18 bit signed multiplier and performs the test expressed by Eq. (15). If the current feature passes the test, a red pixel is sent to the video encoder, otherwise the pixel value from the input stream is transmitted to the video encoder unchanged. The output of the video encoder is connected in turn to a NTSC monitor. As a result, when we connect the input of the video decoder to a camera, we can see in the monitor a red pixel centered in each pixel neighborhood that is selected as a good feature. In the current version of the system and the non-maximum suppression step has not been implemented yet due to lack of logic resources. For this reason, instead of a single red dot corresponding to the peak of  $P_{\lambda_t}(i, j)$ , on the NTSC monitor we actually see a small number of red pixels clustered around the real feature location. The value of the minimum eigenvalue  $\lambda_t$  has been hardcoded into the FPGA's configurations.

## 5 Experimental results and conclusion

We have observed reliable real-time operation (30 Hz) of the system on the NTSC signal generated by a commercial video camera on different scenes and under different light conditions. A MPEG movie, available at <http://vision.caltech.edu/arrigo/movie.mpg> demonstrates the operation of the system in real-time. The total delay between the input and the output pixel streams is the time the decoder takes to scan five lines and 21 pixels, and is equal to 321.5  $\mu$ s. A software implementation of the same method, running on a 100 MHz Pentium PC, takes almost 2 s to process a single frame. The timing reports produced by the CAD tools used to generate the FPGA designs have shown that the bottleneck of this system is in the rate at which the video encoder supplies pixel data to the pipeline. For this reason, we believe that a feature selection system built with present generation FPGA's is able to perform at even higher frame rates. We are currently working on the FPGA/host interface in order to access both the feature coordinates and the image pixel

values, since this will allow us to address the correspondence problem from a software point of view.

The goal of this work was twofold-to solve one of the most challenging problems in computational vision and to investigate the use of arrays of FPGA's for implementing low-level vision tasks in real-time. While mapping the feature selection algorithm onto the G800 architecture, we have learned several lessons on the tradeoffs faced by the designer of a multi-FPGA system. Given that the density of current FPGA's is increasing steadily, we have seen that the fundamental limitation of current and future reconfigurable systems is not in the total amount of logic borne by each FPGA, but in the constraints imposed by interconnection resources.

## 6 Acknowledgments

The authors acknowledge Synopsys and Xilinx for their continued support, and are grateful to Paul Laity for providing us with a pre-release of the Xilinx Logi-Core Multiplier library and to Brad Taylor for many discussions on porting the feature selection algorithm to the G800 reconfigurable computer.

## References

- [1] S. Smith and J. Brady, "Asset-2: Real-time Motion Segmentation and Shape Tracking," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 8, no. 17, pp. 814-820, 1995.
- [2] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *Proceedings of the 4th Alvey Vision Conference*, pp. 147-151, 1988.
- [3] P. Athanas and L. Abbott, "Addressing the Computational Requirements of Image Processing with a Custom Computing Machine: An Overview," in *Proceedings of the 2nd Workshop on Reconfigurable Architectures*, Apr. 1995, Santa Barbara, CA.
- [4] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, Apr. 1991.
- [5] Xilinx, *The Programmable Logic Data Book*, 1996, San Jose, CA.
- [6] Giga Operations Corporation, Berkeley, CA, *Giga Operations Spectrum Documentation*, 1995.
- [7] G. Golub and C. V. Loan, *Matrix Computations*. Johns Hopkins, third ed., 1996.
- [8] Y. Li and W. Chu, "Implementation of Single Precision Floating Point Square Root on FPGAs", *Proceedings of the IEEE Symposium on FPGA's for Custom Computing Machines*, pp. 226-232, Apr. 1997, Napa (CA).
- [9] D. B. (editor), *Splash 2: "FPGA's in a Custom Computing Machine"*. IEEE Computer Society Press, 1996.
- [10] Synopsys, *FPGA Compiler User Guide v3.5*, Sep. 1996, Mountain View, CA.