# Subpixel motion computing architecture

J. Díaz, E. Ros, S. Mota, F. Pelayo and E.M. Ortigosa

**Abstract:** A pipelined optical-flow processing system that works as a virtual motion sensor has been described. It is based on a field programmable gate array (FPGA) device enabling the easy change of configuring parameters to adapt the sensor to different speeds, light conditions and other environmental factors. It is referred to as a 'virtual sensor' because it consists of a conventional camera as front-end supported by an FPGA processing device, which embeds the frame grabber, optical-flow algorithm implementation, output module and some configuration and storage circuitry. This is the first fully stand-alone working optical-flow processing system to include both accuracy and speed of measurement of the platform performance. The customisability of the system for different hardware resources and platforms has also been discussed, showing the resources and performance for a stand-alone board and a PCI co-processing board.

## 1 Introduction

Optical flow computation consists in extracting a dense velocity field from an image sequence assuming that intensity is conserved during displacement. This result may then be used for other applications such as 3-D reconstruction, time interpolation of image sequences, video compression, segmentation from motion, tracking, robot navigation, time-to-collision estimation and so on. The technical problem with estimating the motion of objects in 3-D is that, in the image formation process, because of the perspective projection of the 3-D world onto the 2-D image plane, some of the information is lost. There are several ways of recovering the 3-D information from 2-D images using various cues. These cues are motion, binocular stereopsis, texture, shading and contour. In this paper, we will describe the implementation of a real-time motion flow system, leaving the potential applications for future studies.

Optical-flow algorithms have been widely described in the literature. Some authors have addressed a comparative study of the accuracy of different approaches with synthetic sequences [1]. Their evaluation using real-life sequences is difficult to address because the real optical flow of such sequences is unknown. We have focused on a classical gradient model based on Lucas and Kanade's (L & K) approach [1, 2]. Several authors have emphasised the satisfactory trade-off between accuracy and efficiency in this model, which is an important factor when deciding which model is most suitable to use as a real-time processing system. For a comparative study [1], the L & K algorithm provides very accurate results, added to which, other authors specifically evaluating the efficiency against accuracy trade-off of different optical-flow approaches [3] also regard the L&K model as being quite efficient. Finally, McCane *et al.* [4] also give L & K a good score and conclude that the computational power required by this approach is affordable. This has prompted later researchers to focus on the L&K algorithm [5].

We describe here a hardware implementation of the L&K algorithm. Other authors have recently described the hardware implementation of optical-flow algorithms [6–10], but most of them provide no results to evaluate the performance of the system, that is the accuracy and the computation speed.

The hardware implementation of an algorithm requires a detailed study of the model. After this preliminary stage, simplification strategies are adopted. Finally, it is necessary to evaluate how the adopted simplifications affect the results (Section 5).

We define our system with a high-level hardware description language such as Handel-C [11]. This high-level language enables the easy parameterisation of the design, thus we present a system that can be customised to accomplish different requirements for diverse applications.

## 2 Optical-flow model

Although the original algorithm was proposed as a method to estimate the disparity map in stereo-pair images [2], we have applied Barron's description of the L&K algorithm to optical-flow computation [1]. We have also added several modifications to improve the feasibility of its hardware implementation, we have used IIR temporal filters as described by Fleet and Langley [12] and we have included bias estimations in the detection of maximum gradient when aperture problem appears [13].

In the following equations, we describe briefly the computations upon which the L&K approach is based. A detailed description of the model is provided in previous studies [1, 2].

The algorithm belongs to gradient-based techniques characterised by a gradient search performed on extracted spatial and temporal derivatives. Upon the assumption of constant luminance values through time, the first-order gradient constraint equation (1) is obtained as

$$\nabla_{xy} \boldsymbol{I}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t}) \cdot (\boldsymbol{v}_{\boldsymbol{x}}, \boldsymbol{v}_{\boldsymbol{y}}) + \boldsymbol{I}_{\boldsymbol{t}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t}) = 0 \qquad (1)$$



*IEE Proceedings* online no. 20050207

doi:10.1049/ip-vis:20050207

Paper first received 19th July 2005 and in revised form 23rd June 2006

J. Díaz, E. Ros, F. Pelayo and E.M. Ortigosa are with the Department of Computer Architecture and Technology, University of Granada, Spain

S. Mota is with the Department of Computer Science and Numerical Analysis, University of Cordoba, Spain

E-mail: jdiaz@atc.ugr.es

This equation only allows us to estimate velocity in the direction of maximum gradient, that is in the normal direction of moving surfaces. To overcome this limitation, the L&K method constructs a flow estimation based on the first-order derivatives of the image. By least-square fitting, the model extracts an estimation of motion on the basis of the hypothesis of similarity of velocity values in the neighbourhood of a central pixel. This is described mathematically in (2)

$$\min \sum_{x\in\Omega} W^2(x)[\nabla_{xy} I(x,y,t)\cdot(v_x,v_y)+I_t(x,y,t)]^2 \quad (2)$$

where $W(x)$ weights the constraints with values near the centre of the spatial neighbourhood $\Omega$.

The known solution to this problem is expressed in (3) and (4).

$$\vec{v}=[A^{\mathrm{T}}W^2A]^{-1}A^{\mathrm{T}}W^2\vec{b} \quad (3)$$

where

$$A^{\mathrm{T}}W^2A=\begin{bmatrix}\sum_{x\in\Omega}W^2I_x^2+\alpha & \sum_{x\in\Omega}W^2I_xI_y\\ \sum_{x\in\Omega}W^2I_xI_y & \sum_{x\in\Omega}W^2I_y^2+\alpha\end{bmatrix}$$
$$A^{\mathrm{T}}W^2\vec{b}=\begin{bmatrix}-\sum_{x\in\Omega}W^2I_xI_t\\ -\sum_{x\in\Omega}W^2I_yI_t\end{bmatrix} \quad (4)$$

An inherent limitation to these models appears in blank wall or aperture problem situations. In these cases, the problem has no solution (matrix $A^{\mathrm{T}}W^2A$ is not invertible) and the model cannot provide any estimation of motion. To overcome this, we have added a small constant, $\alpha$, to the matrix diagonal, as suggested by Simoncelli *et al.* [13], which allows us to estimate the normal velocity field in situations where 2-D velocity cannot be extracted because of the lack of contrast information. In summary, we have to compute the $2\times2$ matrix of (4-left), its inverse and the $2\times1$ matrix indicated in (4-right).

Before computing the image derivatives in the matrix elements of (4) they are pre-processed by Gaussian smoothing, which reduces image noise and generates a higher correlation between adjacent pixels. Typically, Gaussian space-time filters of 2 pixels variance plus a temporal derivative of 5 pixels are used. All the temporal operations require storage of 15 images for the entire process. This is hardly affordable in embedded hardware systems. Therefore as indicated in the work of Fleet and Langley [12], a more efficient tactic can be implemented by using IIR temporal recursive smoothing and derivative filters. In this way, the temporal storage requirement is reduced to three frames and the computation time improved at a cost of only slightly reduced accuracy. The temporal filter can be computed as follows.

Let us consider a separable space-time smoothing filter. After the spatial filtering operation, we can use a causal temporal filter based on a truncated exponential.

$$E(t)=\begin{cases}\exp(-t/\tau)/\tau & t\geq 0\\ 0 & t<0\end{cases} \quad (5)$$

where $\tau$ is the time constant of the filter. The temporal derivative of the images can be calculated using this filter. The digital filter equations [12] are

$$\begin{aligned} w(t)&=I(t)-2rw(t-1)-r^2w(t-2)\\ R_2(t)&=q^2w(t)+2q^2w(t-1)+qw(t-2)\\ y(t)&=R_2(t)-ry(t-1)\end{aligned} \quad (6)$$

where we store and update: $w(t-1)$, $w(t-2)$, $y(t-1)$. The parameters $q$ and $r$ are calculated from $\tau$ according to (7)

$$q=\frac{1}{1+2\tau}\quad r=\frac{1-2\tau}{1+2\tau} \quad (7)$$

Finally, the smoothed temporal image and its derivative are computed with (8)

$$\begin{aligned} I_{\mathrm{smooth}}(t)&=qy(t)+qy(t-1)\\ I_t(t)&=\frac{(I_2(t)-I_{\mathrm{smooth}})}{\tau}\end{aligned} \quad (8)$$

## 3 Hardware implementation

Nowadays real-time computation of simple optical-flow algorithms for small images is possible by software because of the outstanding computational power of PCs. The drawback is that it is difficult to adapt these systems to use as embedded solutions, for instance, in robotics applications. Optical-flow can provide to robots navigation information, time-to-contact or tracking capabilities. Current robots incorporate simple motion information extraction schemes [14] that limit unnecessarily the potential applications of the system. In interactive robotic tasks, high temporal resolution is crucial, and this requires high power computation to extract reliable motion information in real-time. The presented approach fulfils this requirement, as the processing is done at conventional frame rates with different spatial resolutions.

For recovering 3-D structure from motion [15], large images, dense information and high accuracy is needed. The huge computational power required for this task makes this application difficult to address with conventional computing platforms.

Encoding standards also use motion information. The actual implementations typically use block-matching methods and try to minimise the total coding error (often measured as signal-to-noise ratio). However, this does not take into account the subjective appearance of the coding artefacts, which can significantly affect the video quality. High-quality motion estimation methods [16, 17] lead to higher quality pictures compared with the use of a simple 'best match' motion estimator.

As it can be seen from earlier discussions, diverse potential applications can benefit from the development of a customisable optical flow system of high computational power and high quality. The customisation feasibility is the key-factor to address several applications with the same technology. The possibilities of working for a specific-application while managing design trade-offs (such as different frame-rate against spatial resolution and customised flow-accuracy against system cost) are very advantageous. The solution we propose is based on the use of programmable logic circuits (FPGAs), where the motion computation chip can be regarded as part of a smart sensor. These circuits allow us to design a customised DSP circuit in a single chip of high computational power based on an intensive use of their intrinsic parallelism and pipeline resources. As we will show in later sections, the solution described here uses this technology to implement a real-time hardware

device capable of working as a PC co-processor or as a smart sensor in embedded applications.

### 3.1 Hardware development

For our design, we have used two platforms: the first one is the RC1000-PP board from Celoxica [18]. This is a PCI bus board connected to the PC and can be used as a hardware accelerator board or as a prototype board. It contains a Virtex 2000E-6 Xilinx FPGA and four 2 MB SRAM memory banks accessible in parallel. This platform uses a client–server scheme. A host program written in C/C++ sends image data to the FPGA through the PCI bus and then receives data processed by the FPGA. Because we are interested in stand-alone hardware devices, all the computations have been made with the FPGA, but this platform is also flexible and capable of working within more complex co-design schemes. The second platform is the stand-alone RC203 board from Celoxica [18]. This board includes camera input, video/VGA output, two 2 MB SSRAM memory banks and a XC2V3000-4 FPGA. It is a very suitable test system for embedded applications.

We have used Handel-C [11] as hardware specification language to generate the Edif input to the Xilinx ISE environment. This high-level hardware language allows us to describe register transfer level (RTL) circuits in a very algorithmic-like way. This is relevant because of the algorithmic nature of the proposed method that makes an RTL approach more difficult to adopt. The drawback is the cost in terms of number of gates, but the design time is reduced significantly. Finally, the Xilinx tool generates the programming file for the FPGA.

### 3.2 System implementation overview

The processing schemes of the PCI co-processing board and the stand-alone platform are illustrated in Figs. 1*a* and *b*. The efficient implementation of the algorithm with an FPGA device requires the intensive exploitation of the intrinsic processing parallelism of this kind of device. We use segmented architecture, as shown in Fig. 1*c*.

The basic computational stages in Fig. 1*c* can be summarised as follows.

- $S_0$. The frame-grabber receives the pixels from the camera and stores them in one of the memory banks, using a double-buffer technique to avoid temporisation problems.
- $S_1$. Spatial-Gaussian-filter smoothing stage.
- $S_2$. The IIR temporal filter computes temporal derivative and space-time smoothed images.
- $S_3$. Spatial derivatives stage.
- $S_4$. Construction of least-square matrices of (4).
- $S_5$. Custom floating-point unit. Final velocity estimation requires the computation of a matrix inversion, which includes a division operation. At this stage, the resolution of the incoming data bits is significant and expensive arithmetic operations are required. Thus fixed-point arithmetic becomes too expensive, prompting us to design a customised floating-point unit.

The computation bit-width increases throughout the pipeline structure. For example, for a high-precision system with low accuracy degradation, we use 8 bits in the first two stages, 12 bits in the third and fourth stages, 24 in the construction of the least-square matrices and 25 for the floating-point unit. The computation of the least-square matrices ($S_4$) is the most expensive stage in terms of computational resources. Different parallelism strategies can be adopted at this point. We have in fact tested a less expensive approach in terms of hardware with good qualitative results.

The basic parameters of the pipeline structure are latency (L) and the maximum number of cycles (MNC) required during the longest stage, which is the limiting factor of the computing speed. The pipeline circuit scheme provides a computing speed (data throughput) in pixels per second (pps) that depends on the MNC and the frequency clock according to the expression $\text{pps} = f_{\text{clk}}/\text{MNC}$.

*3.2.1 Bit-width estimation:* The bit-width at each stage is a very important parameter because it significantly

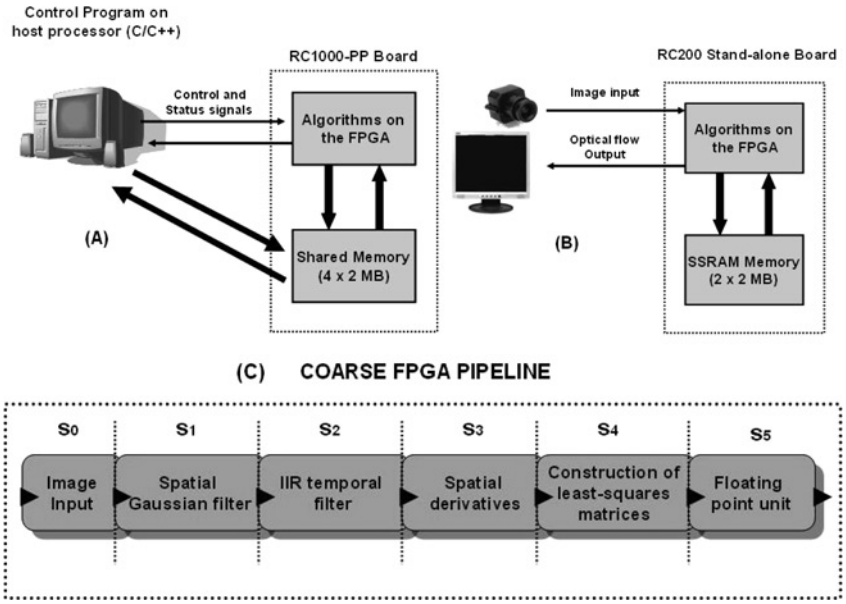


**Fig. 1** *Boards schemes and FPGA pipeline architecture*

*a* PCI-board scheme
*b* Stand-alone board scheme
*c* Coarse pipeline processing architecture for optical flow computation

affects system quality and hardware requirements. Several decisions have to be arrived at:

1. The data bit-width for data registers.
2. Arithmetic data representation: integer, fixed point or floating point.
3. Rescaling data or data range compression after operations: by truncation, wrapping or nonlinear range compression such as saturation or logarithmic data range compression.

All the design decisions summarised above have been taken after several trials to obtain a balanced design in which the degradation of the results along the data-path is minimised and hardware resources are kept to affordable levels. The following considerations have been taken into account.

1. Convolution operations can be implemented efficiently with integer or fixed-point arithmetic.
2. Operations such as division or multiplication with high bit-widths are more efficiently implemented with floating-point arithmetic. This is necessary to process data with reasonable bit-width precision during some stages.
3. Nonlinear range compression such as logarithmic data range compression (for rescaling data after arithmetic operations) is not well suited to FPGA devices because of its complex implementation, unless lookup tables are used. The effects on system quality have not been studied yet, so we use truncation and wrapping techniques. A statistical study can be made to evaluate how this operation affects the quality of the results.

### 3.3 Critical stages

There are two main critical stages: $S_4$ and $S_5$. The construction of least-square matrices is done in $S_4$ where the trade-off between efficiency and cost can vary widely. Equation (4) requires the computation of five products: $\boldsymbol{I}_x^2$, $\boldsymbol{I}_y^2$, $\boldsymbol{I}_x\boldsymbol{I}_y$, $\boldsymbol{I}_x\boldsymbol{I}_t$, $\boldsymbol{I}_y\boldsymbol{I}_t$. Thus we make a weighted sum in a window ($\Omega$) over a neighbourhood of size $\boldsymbol{w}_x \times \boldsymbol{w}_y$. Owing to memory limitations, we save the $\boldsymbol{I}_x$, $\boldsymbol{I}_y$ and $\boldsymbol{I}_t$ values instead of the five crossed products. Therefore the operations made are:

- computation of the products for all the elements within a neighbourhood. We need to calculate five $\boldsymbol{w}_x \times \boldsymbol{w}_y$ multiplications;
- row-convolution operation. We compute five multiply by $\boldsymbol{w}_y$ convolutions;
- column-convolution operation, requiring the computation of five convolutions.

This is an important stage where we can bias the trade-off between efficiency and hardware cost. For example, if we use a $3 \times 3$ neighbourhood, we need between 1 to 45 multipliers, 1 to 15 row-convolution units and 1 to 5 column-convolution units. This choice allows us to compute the weighted sum values in one clock cycle with a highly parallel hardware unit or to compute it sequentially.

This has been schematically represented in Fig. 2 for three implementations with different levels of parallelism and with an integration area of $3 \times 3$ for the least-squares neighbourhood. Fig. 2*a* represents a very parallel datapath that we call high-speed (HS) version and which achieves MNC = 10 cycles. A slower version was implemented with less parallelism (reducing the number of parallel multipliers and row/column convolution units) as shown in Fig. 2*b*, thus resulting in a medium speed (MS) version. This configuration leads to a limiting data throughput stage with MNC = 26. Finally, we implemented a low-speed (LS) version represented in Fig. 2*c* with just one column and one row convolution unit and MNC = 42.

The second critical stage is the computation of final velocities using a custom floating point unit. At this stage, (3) is computed. Until now, the arithmetic operations have been done using integer or fixed-point arithmetic with truncation operations. Convolution operations work well with this representation but when bit-width is too large, a floating-point representation of the data is better suited for hardware implementation. This is done with a customised superscalar floating-point unit. As during the previous stage ($S_4$), a high bit-width (24 bits) is used to preserve computational accuracy, the current stage ($S_5$) becomes very expensive in terms of hardware resources. Therefore the design of $S_5$ is critical as it exerts an important influence on the accuracy against processing speed trade-off.

The calculations in this stage involve the following basic arithmetical operations: subtraction, multiplication and division. When arithmetical operations are made with high bit-width, the signal delays associated with carry lines degrade overall performance, decreasing the maximum system frequency. To avoid this, pipeline arithmetic operators or sequential iterative operators can be used. The first approach allows us to make the computation in 1 or 2 clock cycles, after a given latency at a high cost in terms of hardware resources. The second option takes several clock cycles therefore degrading the MNC of the system, but allows us to use the same hardware for each iteration. We define a system that uses one-cycle floating-point hardware circuits, because this works at the desired maximum clock frequency (without becoming the limiting stage) for all the operations except the division. We have used a hardware sequential divisor instead of a pipelined divisor that needs 21 cycles to compute the division of 25 bits of floating numbers. But in this case, the MNC is too high and imposes a considerable limit on pipeline performance. To counter this, we use up to three-way division units and, depending on the performance required, we can synthesise more or less ways. Each floating-point unit needs:

1. one to five fixed-point to floating-point converter units;
2. one to six 25-bit floating point multipliers;
3. one to three subtractors;
4. one or two divisor units. If an $n$-ways divisor scheme is chosen, we use $n$ to $2n$ divisor units.

The hardware consumption resources using different configurations are represented in Fig. 3. More concretely, in Fig. 3*a*, we consider a high parallel implementation with several multipliers, adders, division units as well as fixed to floating point converters (all these are used in the HS version). Fig. 3*b* represents an implementation with less parallelism used for MS and LS versions. The high MNC values of MS and LS versions (using a limited parallelism level) allow a high degree of resources sharing.

## 4 Hardware performance and resources consumption study

The system is designed in modules, so that parallelism and bit accuracy at different stages can be easily modified. Owing to the high level of abstraction that Handel-C provides [11], it is easy to manage the parallelism of the computing circuits and the bit-width at the different stages.

One important aspect is that of the various possibilities for configuring the system. We have evaluated several configurations to explore different trade-offs between accuracy,

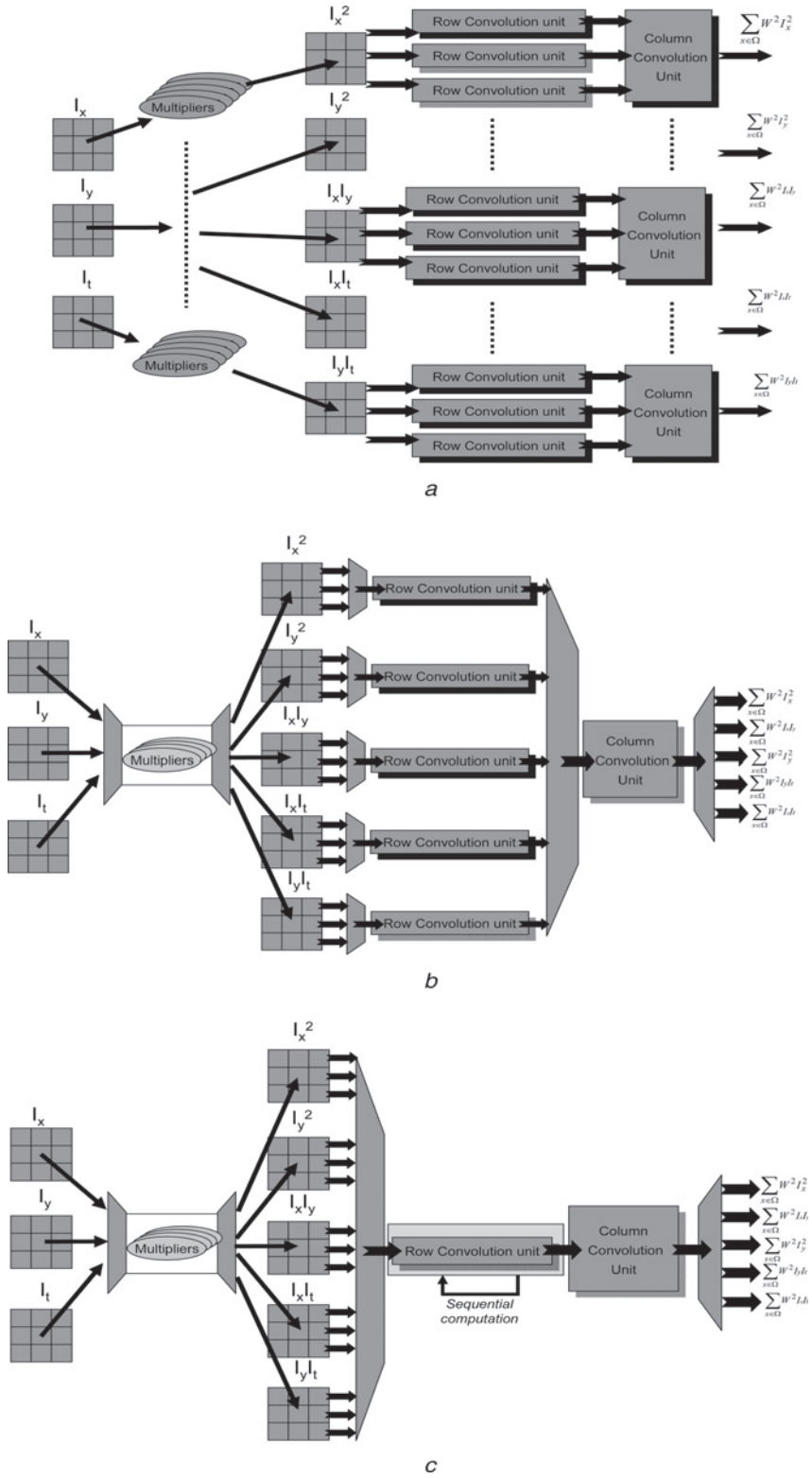


**Fig. 2** *Stage $S_4$ architecture, parallelism levels of the different versions are schematically shown*

*a* Architecture of the HS version ($S_4$).
Note the large number of parallel scalar units and multipliers that are used
*b* Architecture for the MS version ($S_4$).
Row convolution units are shared for each derivative product and only one group of multipliers is used
*c* Architecture for the LS version ($S_4$)
Only one group of multipliers, one row and one column of convolution units are used for this version

hardware cost and computing speed. In all these configurations, we have used the same basic architecture but with different levels of parallelism as shown in Figs. 2 and 3. Table 1 summarises the main properties of the different configurations. The one using a 5 × 5 average window for the least-square-matrix neighbourhood is the one we have called high-quality (HQ) approach, and the ones using a 3 × 3 window, medium quality (MQ). Other modifiable parameters are the smoothing and spatial derivative filter sizes. HQ and MQ approaches include 5-pixel derivative

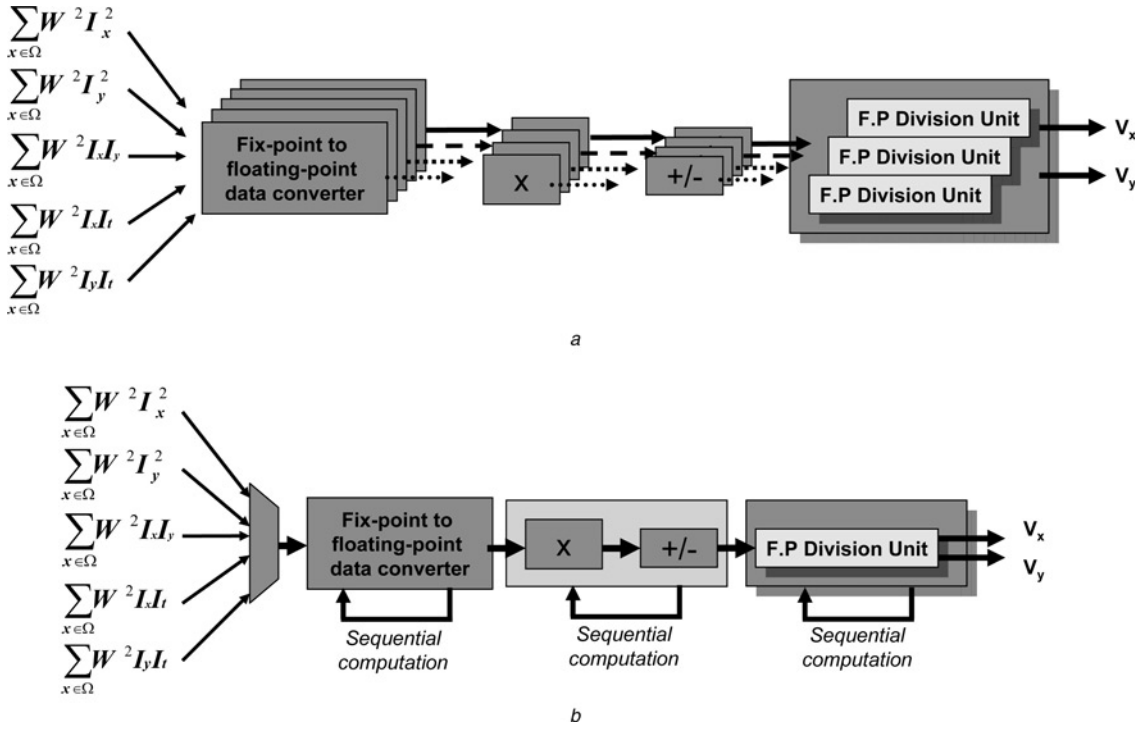


**Fig. 3** *Architecture description of stage $S_5$, different numbers of scalar units are used depending on the target parallelism level*

*a* HS version with a larger number of parallel scalar units
*b* MS and LS versions
MNC value of the previous stage $S_4$ allows sharing most of the processing units used at this stage as well as the fixed point to floating point converters, multipliers adders or the division unit which only requires one way

filters and 9-pixel Gaussians. In a more economical way, the low cost (LQ) version uses 3-pixel Gaussian and derivatives filters.

If we fix the optical-flow quality of the system, another factor to take into account is the performance against hardware cost trade-off. As commented in Section 3.3, we have designed datapaths with different levels of parallelism and which lead to different performances (i.e. diverse MNC values). Table 1 summarises the performance of the systems and hardware costs.

It is important to note that in our experiments data transmission of the images to the prototyping board through the 33 MHz PCI bus takes about 30–40% of the total processing time and therefore higher frame rates might be expected using a direct connection between the camera and the FPGA. Furthermore, as explained in Section 1, the data-throughput of the HSHQ is 2.700 Kpps at 27 MHz of clock frequency (in Table 1 this is limited to 1776 by the PCI bandwidth). This topic is discussed in the work of Benitez [19].

We have tested the design using the stand-alone prototyping platform RC203 from Celoxica [18] to avoid the PCI bus bottleneck. This platform includes an XC2V3000-4 FPGA with embedded multipliers. In this approach, we have implemented the whole optical flow system plus Video input, VGA and a memory arbitration controller. A LUT for visual colour representation of the velocities vector for the VGA output has also been included in the FPGA. The optical flow system implemented is based on the HSHQ version but uses the embedded multipliers and several clock domains. Table 2 shows the hardware costs and the system performance.

The computing speed measured with $f_{clk} = 41$ MHz was 4100 kpps (53 fps of 340 × 280 images, MNC = 10). Now the system is faster due to the improved technology of the Virtex II. The use of a customisable approach with a high-level description language facilitates the implementation of this system on different platforms. In fact, the optical flow processing algorithm only consumes 80% of the number of slices while the rest is occupied by the I/O controllers.

**Table 1: Performance and hardware cost of different configurations on a Virtex 2000-E FPGA (2 million gates, 43 200 logical cells (LC) and 640 Kbits of embedded memory)**

| Version | % device occupation | % on-chip memory | Kpps | Image resolution | Fps, $f_{clk} = 27$ MHz | Maximum $f_{clk}$, MHz |
|---|---|---|---|---|---|---|
| HSHQ | 99 | 17 | 1776 | 160 × 120 | 95 | 35 |
| | | 31 | | 320 × 240 | 24 | |
| HSMQ | 65 | 16 | 1776 | 160 × 120 | 97 | 35 |
| | | 31 | | 320 × 240 | 24 | |
| MSMQ | 43 | 16 | 625 | 160 × 120 | 33 | 35 |
| LSLQ | 36 | 8 | 400 | 120 × 90 | 38 | 35 |

K, ×1000; Kpps, kilopixels per second; Fps, frames per second

**Table 2: Performance and hardware costs of a stand-alone system with camera input and VGA output on a Virtex II XC2V3000-4 FPGA (3 million gates and 1728 Kbits of embedded memory)**

| Version | % device occupation | % on-chip memory | % embedded multipliers | Kpps | Image resolution | Fps | Maximum $f_{clk}$, MHz |
|---|---|---|---|---|---|---|---|
| Stand-alone RC203 board | 99 | 29 | 41 | 4100 | 340 × 280 | 53 | 41 |

The use of the embedded multipliers saves 1324 LCs (about the 4.6% of the resources of the device).

A critical design issue is the internal and external memory management of our system. The communication bandwidth shall be high enough not to degrade the computing performance. Internal memory [called embedded memory blocks (EMBs)] can be arranged with a high degree of flexibility, allowing a high-communication bandwidth but only with very small capacity (640 Kbits for the device used in the PCI board). Owing to this fact, we have only used this resource for the internal convolution units designed in our system and synchronisation buffers. For a 320 × 240 image resolution, 44 EMBs are required in the HS version, 32 in the MS approach and 24 in the LS implementation. Increasing the image resolution means obtaining longer rows. For example, a VGA image resolution requires multiplying by two the number of EMBs, but this is not a problem because we have plenty of EMBs available.

The real bottleneck may be the external memory access. In our system, the PCI board version includes four parallel memory banks with 4-byte memory words each. We have designed an efficient memory management unit with an MNC = 1 that allows reading or writing these 4 bytes in just one clock cycle. This produces a memory bandwidth of 16 bytes per clock cycle, which completely fulfils our requirements (we require ten accesses in ten cycles). Furthermore, this efficient management of the external memory resources is a key factor that allows using just the only two memory blocks available in the stand-alone system or replicating the number of processing cores if further performance is required.

We have shown the system's flexibility and the trade-off between the number of gates and performance. Another important subject is that of scalability at the level of functional units. All our results work on the assumption that only one computational unit is used. A local image processing algorithm can take advantage of the possibility of splitting the FPGA. We can synthesise some computational units in the same FPGA or in several of them and compute larger images in real-time. If a memory buffer is used, it is straightforward to assign a small area to each computational unit and run it in parallel. The computational power is then increased by a factor close to the number of computational units running in parallel. See a more detailed example in Section 6.

### 4.1 Comparison with other approaches

The implementation of the optical-flow algorithm with FPGA has only been addressed by some authors in very recent years. Table 3 summarises the results from different approaches. On the basis of iterative algorithm of Horn and Schunk (H&S) [20], Martin *et al.* [10] makes a system implementation that fits quite well the specification of an optical flow system but just simulation results are presented. The main disadvantage of this approach is that performance of H&S model is poor compared with other alternatives [1]. For example, the evaluation of the Yosemite sequence with

**Table 3: Comparison with prior approaches**

| Implementation | Throughput, Kpixels/s | Image size, pixels | Image rate, frames/s |
|---|---|---|---|
| L&K stand-alone board (described here) | 4100 | 320 × 240 | 53 |
| L&K PCI-board ([22] and described here) | 1776 | 320 × 240 | 30 |
| H&S [10] | 3932 | 256 × 256 | 60 |
| Block-matching [6] | 9216 | 640 × 480 | 30 |
| L&K [9] | 1692 | 256 × 256 | 26 |
| H&S [21] | 47 | 50 × 50 | 19 |

Our data has been obtained using the maximum available system clock frequency

the L&K method leads to results with 4.28° of average errors (density 35.1%), whereas with the Barron improved version of H&S, taking more than 100 iterations, the average error is 5.59° (density 32.9). This represents a relative error 23% higher than the L&K algorithm adopted in the approach presented here considering that the density is similar. This is also the case in Cobos and Monasterio [21], which uses the same algorithm but addressing an implementation with less resources and therefore with worse performance. Using the block-matching approach, the implementation described by Niitsuma and Maruyama [6] achieves 30 fps of image size 640 × 480 but with high hardware cost (90% slices of a XC2V6000 FPGA) and without sub-pixel accuracy. Another study has been published recently [8], in which the hardware requirements are lower but no information about the system performance is provided.

On the basis of the L&K approach, Correia and Campilho [9] recently presented a real-time implementation of the system using a MaxVideo200 pipeline image processor. With this accelerator board, the performance and accuracy are similar to our results (we obtained a maximum performance of 1776 Kpps with our PCI board system, 4100 Kpps with the stand-alone, whereas they obtained 1692 Kpps). However, the use of an acceleration processor makes it difficult to be transferred to embedded applications. Finally, the model described here, running in software on an AMD 1800 + MHz, can compute 1666 Kpps (for instance arranged in 25 fps of 160 × 120 pixels) and this could be optimised using MMX and SSE instructions. The problem is that in this case it consumes all the computing resources of the machine.

## 5 System accuracy evaluation

As commented in the introduction, the accuracy of the computation of the optical flow in real-life sequences is difficult to assess because the real flow of these sequences is unknown. Therefore to evaluate the accuracy of our design, which depends on the bit-width of the different stages, we have adopted two different evaluation

methods. First, in Section 5.1, we use the test scheme and a synthetic sequence from the comparative study made by Barron *et al.* [1], with the error measurement proposed by Fleet and Jepson [23]. This error measurement has been widely used in the literature, being therefore appropriate to compare our results with previous works. This measurement can be used with high- and low-velocity modules with the same estimators but with some bias. A more detailed explanation about this issue can be found in Barron *et al.* [1]. Secondly, Section 5.2 shows some qualitative results that illustrate the accuracy of our system.

### 5.1 Synthetic sequences with known optical flow

In the hardware implementation some simplifications are made to the original model. Table 4 shows the accuracy of the model after the modification of several parameters. Unthresholded results (100% density) are included to enable an easy comparison between the hardware and software versions. The second row in Table 4 includes the evaluation results with reliable estimations as indicated in Barron *et al.* [1].

The fourth and fifth rows include results of the implementation of the algorithm with IIR filters computed with fixed point arithmetic using 12 bit-width. In the sixth row of Table 4, the accuracy of the L&K algorithm (with hardware-system parameters) is computed by a standard PC using double precision variables and unthresholded results. Finally, the seventh row includes the performance achieved with our hardware implementation. It can be seen that accuracy is reasonably high, bearing in mind that fixed-point variables and restricted bit-widths are used in this approach. It can be seen that the performance of the hardware is only slightly worse (2.48° increase in error) than the software version with a data precision of 64 bits. Furthermore, the results of the hardware implementation described here are comparable with other software approaches evaluated by Barron *et al.* [1].

We have also compared the performance of the software and the hardware implementations using sinusoidal grating sequences. We used different stimulus frequencies ($f_0 = 0.02$ and $f_0 = 0.05$) and velocities ($V = 0.25$ ppf and $V = 1$ ppf). With these tests, the hardware achieved results very similar to those of the software (less than 5% error in the speed of calculation).

### 5.2 Real sequences: qualitative optical flow results

Only qualitative differences were estimated with both the hardware and software optical-flow approaches using real sequences (as the real flow is unknown). In this section, we include some real-image sequences for a qualitative evaluation.

Fig. 4 contains the image of an overtaking-car sequence, together with the results of software (LK IIR version) and hardware optical-flow estimations. This is a good example of how optical flow can be used for certain real-life applications in a very straightforward way. In the example, the goal is the segmentation of the overtaking car, which can easily be done relying on optical flow, as the motion pattern of the overtaking car (moving rightward in the images) contrasts sharply with the landmarks, moving leftwards due to the egomotion of the host car.

As shown in Figs. 4*b* and *c*, the software results are smoother than those produced by the hardware. This is due to the Bit-width restriction of the hardware approach. Nevertheless, the results are quite similar and the accuracy of the hardware seems to be enough to obtain good qualitative results.

Fig. 5 shows illustrative results using the different version qualities (HQ, MQ and LQ) also in the framework of the rear-view mirror (refer to Section 6 for a numerical data evaluation with synthetic data). Note that there is no significant quality degradation for MQ with respect to HQ though the latter produces smoother results. Software results are also quite similar to the hardware ones, which also validate our bit-width decisions. LQ version leads to larger errors but requires only very moderate computing resources.

## 6 System cost evaluation: accuracy against performance trade-off

The comparison of the different system versions requires defining some homogeneous resources utilisation metric that takes into account logic and embedded memory. We will focus on the PCI board version because only results for the HSHQ are presented for the stand-alone system. Table 1 provides detailed information about the resources consumption for the different approaches using percentage of occupation of LC and embedded memory. This information can be easily converted to system gates as in the work of Ortigosa *et al.* [24]. Each LC is equivalent to 12

**Table 4: Yosemite sequence results using the angle error measurement of Fleet and Jepson [23]**

| Model | Average error, degree | Standard deviation, degree | Density, % | Parameters |
|---|---|---|---|---|
| LK FIR | 11.29 | 17.72 | 100 | $\lambda_{min} = 0$, $\alpha = 0$, $\sigma_{xyt} = 1.5$ |
| LK FIR | 4.54 | 11.31 | 33.3 | $\lambda_{min} = 0.75$, $\alpha = 0$, $\sigma_{xyt} = 1.5$ |
| LK IIR | 11.97 | 16.027 | 100 | $\lambda_{min} = 0$, $\sigma_{xy} = 1.5$, $\tau = 2$, $\alpha = 0$ |
| LK IIR (with hardwarised filters) | 11.47 | 15.34 | 100 | $\lambda_{min} = 0$, $\sigma_{xy} = 1.5$, $\tau = 2$, $\alpha = 0$ |
| LK IIR (with hardwarised filters) | 13.71 | 15.99 | 100 | $\lambda_{min} = 0$, $\sigma_{xy} = 1.5$, $\tau = 2$, $\alpha = 1/16$ |
| LK IIR (version implemented in hardware) | 15.91° | 11.5° | 100 | $\lambda_{min} = 0$, $\sigma_{xy} = 0.8$, $\tau = 2$, $\alpha = 1$ |
| Hardware system | 18.30° | 15.8° | 100 | $\lambda_{min} = 0$, $\sigma_{xy} = 0.8$, $\tau = 2$, $\alpha = 1$ |

Comparison between software models (including FIR and IIR filters, and computed with double precision variables) with different parameters [1, 12]. Final row also includes the hardware system accuracy. The fourth, fifth and sixth rows use the simplifications adopted in the hardware implementation (these simplifications are described in Section 3.)

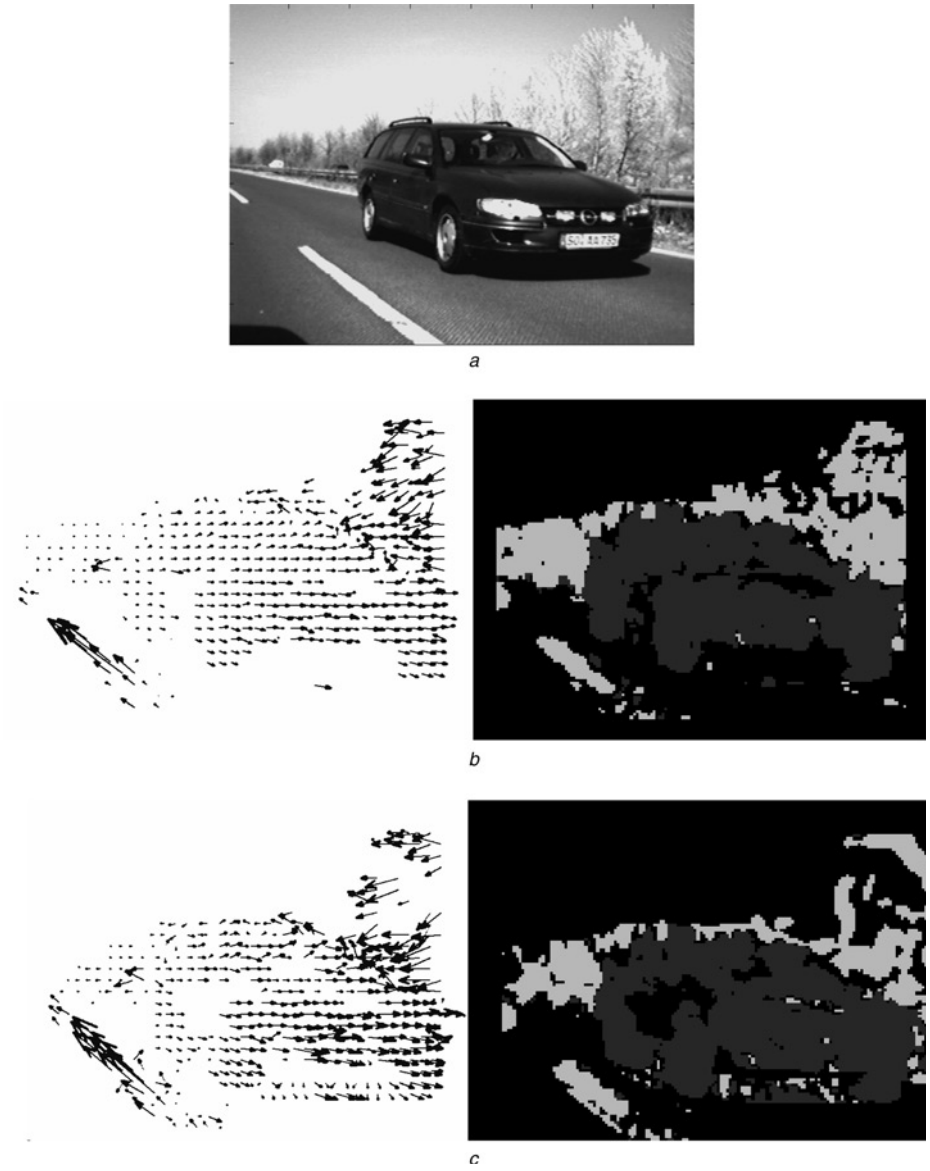

**Fig. 4** *Optical flow for the overtaking car, software against hardware estimations*

*a* Original image extracted from the sequence
*b* Software result
*c* Hardware result
Left-hand images use arrows to represent velocity vectors
In the right-hand images, for the sake of clarity, only leftwards (light shades) due to the landscape and rightwards (dark shades) due to the overtaking-car are used to indicate the motion
From this information the car segmentation is straight-forward

system gates and each memory bit to four system gates. This allows translating the resources consumption into system gates [24]. This facilitates the analysis of the different versions and the generalisation of the results. The system gates estimated for the different versions of Table 1 are presented in Table 5. This metric allows us to define the performance cost ($P_c$) and accuracy cost ($A_c$) as described by (9) and (10), respectively.

$$P_c = \frac{\text{system gates}}{\text{Data throughput}} \tag{9}$$

$$A_c = \beta \cdot \frac{\text{system gates}}{1/(\text{angular error})} \tag{10}$$

where $\beta$ is normalisation constant to scale the $A_c$ to the same range as $P_c$ to facilitate results comparison (we have used $\beta = 175$ for our data).

We have used these parameters to compare the different system alternatives. The results regarding performance and accuracy costs are plotted in Fig. 6. Fig. 6*a* illustrates that the approach with better trade-off between resources and performance is the HSMQ. This is so because it is very parallel but with a reduced least-squares neighbourhood of 3 × 3 pixels. Fig. 6*b* shows that the best approach in terms of accuracy cost is the MSMQ version, with similar cost for the HSMQ and LSLQ.

We can also combine these two performance parameters and define the system cost ($S_c$) as described in (11).

$$S_c = \alpha \cdot P_c + (1 - \alpha) \cdot A_c \tag{11}$$

where $\alpha$ is a parameter between 0 and 1 that weights the different cost contributions. Depending on our target application, we can use different values for this parameter as represented in Fig. 7. We shall take into account that our goal is to minimise the required system gates and maximise

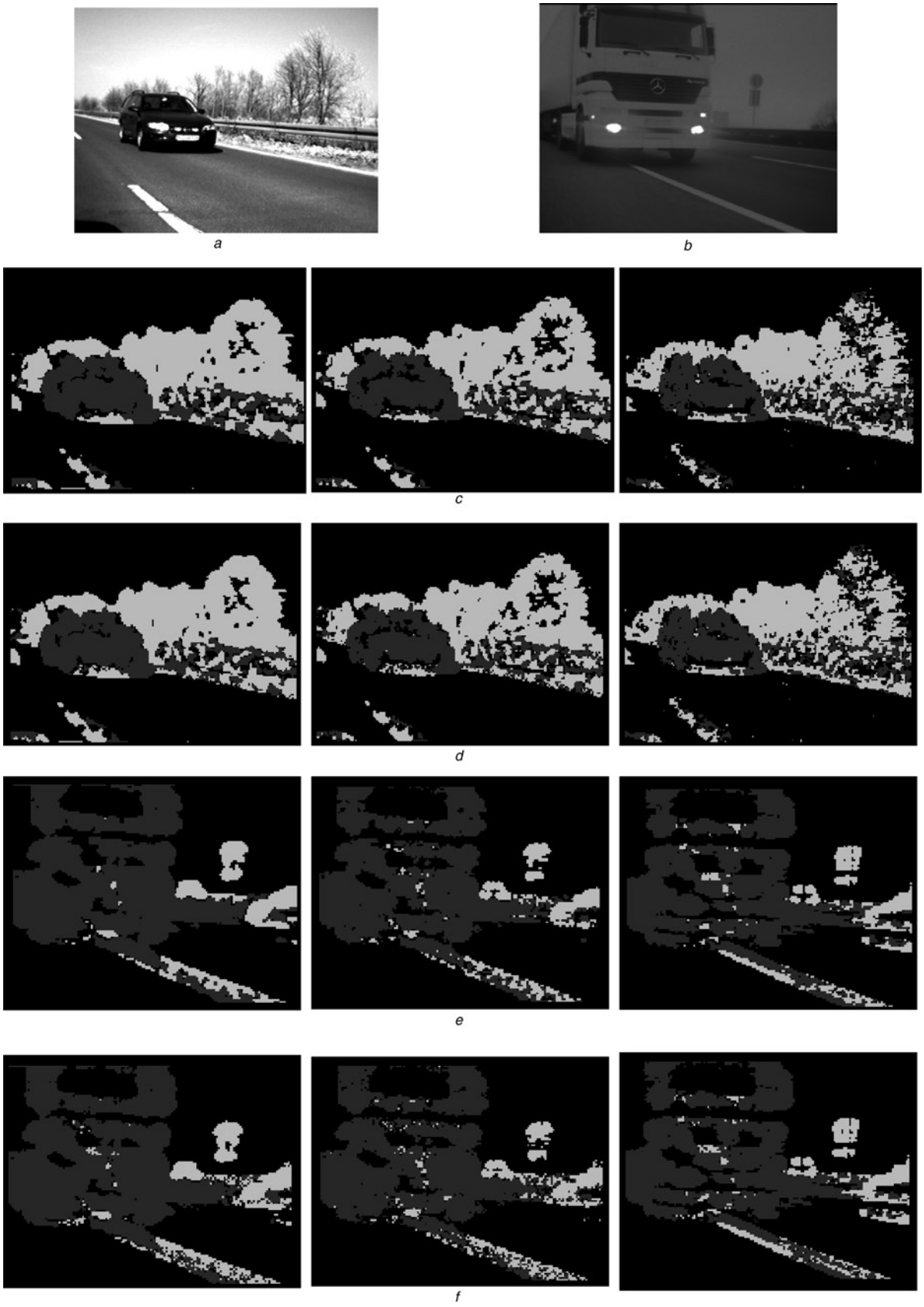

**Fig. 5** *Optical flow results of overtaking sequences*

*a* Car
*b* Truck
Motion is represented with only leftwards (light shades) and rightwards (dark shades), but speed is not presented to facilitate image interpretation
*c* and *e*, respectively, present their optical flow computed with software (double floating point data representation)
*d* and *f* show the results computed with the presented hardware
*c*, *d*, *e* and *f* show the different quality versions (left to right: HQ, MQ, LQ)
Note that, configuring the hardware and software by using the same parameters leads to very similar results
Qualitatively, we see that HQ version is smoother than MQ and LQ versions, which is explained due to the larger least squares neighbourhood
Accuracy evaluation cannot be carried out on these images (as the ground truth is unknown), and because of this fact, we have numerically calculated it with synthetic images, being these results presented in Table 5

**Table 5: Analysis of the performance cost $P_c$ and accuracy cost $A_c$ of the different system versions described in Sections 3.2 and 3.3**

| | System gates, Kgates | Performance, Kpps | Performance cost, gates/pps | Mean error, degree | Accuracy cost, gates · degree |
|---|---|---|---|---|---|
| HSHQ | 1234 | 1776 | 0.69 | 15.149 | 1.79 |
| HSMQ | 861 | 1776 | 0.48 | 16.17 | 1.33 |
| MSMQ | 747 | 625 | 1.2 | 16.17 | 1.16 |
| LSLQ | 580 | 400 | 1.45 | 23.33 | 1.3 |

System gates count the FPGA logic dedicated to processing operations and the embedded memory utilisation. The accuracy of the different system versions is also shown in this table considering an optical flow density of the hardware version of 91%. Note that in terms of accuracy there are no significant differences between HQ and MQ versions, but the error value significantly increases in the LQ approach

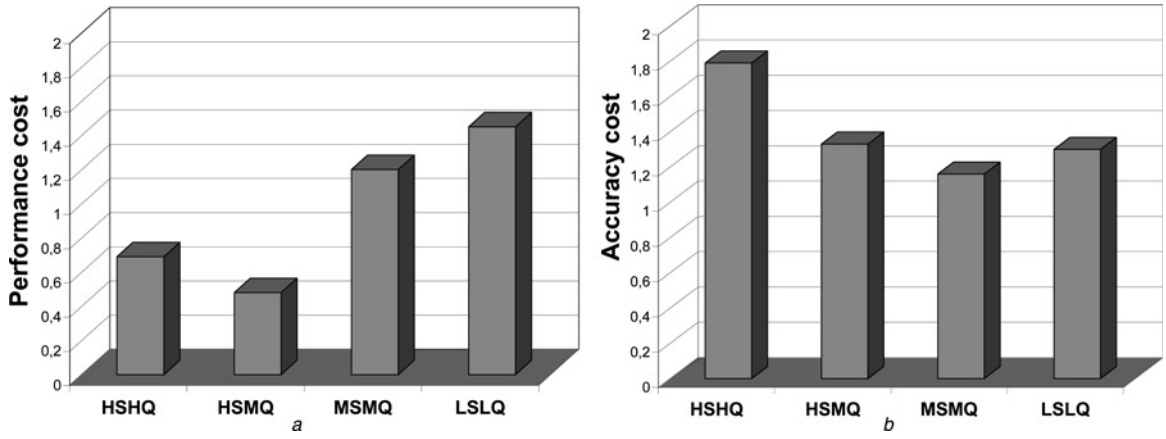


**Fig. 6** *Performance and accuracy costs of different system gates*

*a* Performance cost
*b* Accuracy cost
From these figures, HSMQ and MSMQ are respectively highlighted as the approaches with best accuracy against performance trade-off

performance and accuracy. Fig. 7 shows that the minimum is obtained in most cases for the HSMQ version, which represents the best trade-off between resources consumption, accuracy and performance. The HSHQ approach demands high resources consumption, and therefore is not the best choice for a low-cost device. In contrast, LSLQ produces low accuracy therefore not being a good choice for high precision systems. Finally, MSMQ is the best option if we focus on reducing the accuracy cost.

Though the previous analysis presents the HSMQ version as the best trade-off version, the final decision relies on the target accuracy and on the available resources.

After this comparison, we can evaluate the required resources for a real-time VGA computing performance using the HSMQ version. A simple core replication technique can be used to increase the computing performance as discussed in Section 4. A real-time VGA resolution version requires computing $640 * 480 * 25 = 7680$ Kpps. Our PCI system is able to process up to 1776 Kpps and therefore it is necessary to increase the resources by a factor of 4.3 in order to achieve the target performance. This is a significant increment, which makes this approach hardly affordable, but the main limitation is imposed by the PCI data transmission that consumes about the 35% of the processing time. Current PCI transmission rates (using PCI-Express or PCI-X) are more than four times faster than the used PCI version and allow increasing the performance in a factor of 27%. This can be combined with increasing the system clock frequency. Our design implemented in the Virtex II FPGA family is 17% faster than Virtex-E (as shown in the maximum clock rates included in Tables 1 and 2). These modifications increase the global performance 43% and thus, we only need to replicate the processing core three times.

The stand-alone version does not suffer from the data transmission limitation. Its performance is 41 000 Kpps

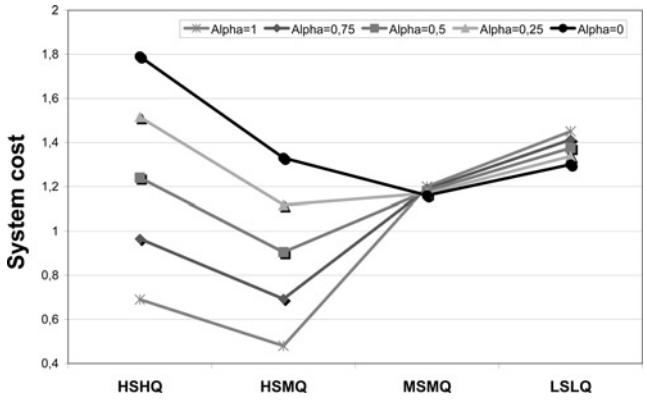


**Fig. 7** *System cost representation for the different system alternatives*

Each line represents different values of $\alpha$ which weights the performance and accuracy costs
Extreme cases are $\alpha = 1$ (optimises only performance cost) and $\alpha = 0$ (focusing only on accuracy cost)
Most of the cases show that the minimum is achieved by the HSMQ version, and therefore this should be pointed out as the version with the best performance against accuracy cost trade-off

and allows achieving real-time VGA resolution at 26 fps with just two cores. This is affordable and fits in current FPGA devices such as the XC2V6000 Virtex II, whereas by being a stand-alone approach it can be used for mobile applications such as robotics or embedded sensors too.

## 7 Conclusions and future work

Reconfigurable computing is a very active research field, which architectures and design methods evolve fast [25]. The system described here shows how an optical-flow estimation circuit can be implemented as a customised DSP system using this technology. The paper describes a scalable architecture that can work with large image data at a conventional video-frame rate (30 fps). System performance, customisation feasibility and scalability, due to the FPGA technology and design strategy, allow the use of the system in diverse application fields as explained in previous sections.

The accuracy of the estimated flow is essential for some of the possible applications outlined in this paper. The results of the hardware implementation are in the range of other software approaches considered in the study of Barron *et al.* [1]. Therefore the performance of the hardware is of reasonable quality provided that it computes in real-time (at a speed of 1776 Kpps with our PCI board system at 27 MHz, 4100 Kpps with the stand-alone platform). These results outperform in accuracy and computing previous power approaches.

The described computing platform can be fully implemented on a single chip (as a System-on-Chip), requiring only external memory support for temporal partial results. This has been implemented on an FPGA device efficiently exploiting its inherent computing parallelism and pipeline resources. Furthermore, the modularity of the system enables the easy alteration of the computing scheme to target different computing speed against hardware cost trade-offs.

We have studied how the restricted bit-width of the different computations affects the quality of the extracted optic flow and compared the results obtained with software implementations of the algorithm computed with double precision.

In the future, we plan to apply the described approach in different applications such as tracking systems, robot navigation, video compression and so on. We will evaluate the system requirements for these applications. We will also explore the implementation of multiscale approaches to obtain more reliable flow for different velocity scales. Finally, we also plan to further extend the parallelism level, decreasing the MNC values by adopting a fine-grain pipeline design strategy to increase the computing performance. This will make feasible the utilisation of this kind of specific purpose computing circuits with high frame-rate cameras to increase the optical flow accuracy.

## 8 Acknowledgments

This work has been supported by the V EU research framework funds through the European Projects DRIVSCO (IST-016276-2), SENSOPAC (IST-028056) and the National Spanish Grant DEPROVI (DPI2004-07032).

## 9 References

1 Barron, J.L., Fleet, D.J., and Beauchemin, S.: 'Performance of optical-flow techniques', *Int. J. Comput. Vis.*, 1994, **12**, (1) pp. 43–77
2 Lucas, B.D., and Kanade, T.: 'An iterative image registration technique with an application to stereo vision'. Proc. DARPA Image Understanding Workshop, April 1984, pp. 121–130
3 Liu, H.C., Hong, T.S., Herman, M., Camus, T., and Chellappa, R.: 'Accuracy vs efficiency trade-offs in optical flow algorithms', *Comput. Vis. Image Underst.*, 1998, **72**, (3), pp. 271–286
4 McCane, B., Novins, K., Crannitch, D., and Galvin, B.: 'On benchmarking optical flow', *Comput. Vis. Image Underst.*, 2001, **84**, pp. 126–143
5 Baker, S., and Matthews, I.: 'Lucas–Kanade 20 Years on: a unifying framework', *Int. J. Comput. Vis.*, 2004, **56**, (3), pp. 221–255
6 Niitsuma, H., and Maruyama, T.: 'Real-time detection of moving objects', *Lect. Notes Comput. Sci.*, 2004, **3203**, pp. 1153–1157
7 Cobos, P., and Monasterio, F.: 'FPGA implementation of Camus correlation optical flow algorithm for real time images'. Proc. and 14th Int. Conf. on Vision Interface (VI2001), 2001, pp. 32–38
8 Maya-Rueda, S., and Arias-Estrada, M.: 'FPGA processor for real-time optical flow computation', *Lect. Notes Comput. Sci.*, 2003, **2778**, pp. 1103–1016
9 Correia, M.V., and Campilho, A.C.: 'Real-time implementation of an optical flow algorithm'. Int. Conf. on Pattern Recognition (ICPR2002), 2002, pp. 247–250
10 Martín, J.L., Zuloaga, A., Cuadrado, C., Lázaro, J., and Bidarte, U.: 'Hardware implementation of optical flow constraint equation using FPGAs'. CVIU(98), June 2005, vol. 3, pp. 462–490
11 Celoxica Limited.: 'Handel-C language reference manual' (dk2.0 Edn, 2003), RM-1003-4.0
12 Fleet, D.J., and Langley, K.: 'Recursive filters for optical flow', *IEEE Trans. Pattern Anal. Mach. Intell.*, 1995, **17**, (1), pp. 61–67
13 Simoncelli, E.P., and Adelson, E.H., and Heeger, D.J.: 'Probability distributions of optical flow'. IEEE Conf. on Computer Vision and Pattern Recognition, Mauii, Hawaii, June 1991, pp. 310–315
14 Yamada, H., Tominaga, T., and Ichikawa, M.: 'An autonomous flying object navigated by real-time optical flow and visual target detection'. Proc. IEEE Int. Conf. on Field-Programmable Technology, 2003, pp. 222–227
15 Jebara, T., Azarbayejani, A., and Pentland, A.: '3D structure from 2D motion', *IEEE Signal Process. Mag.*, 1999, **16**, (3), pp. 66–84
16 Thomas, G.A., and Dancer, S.J.: 'Improved motion estimation for MPEG coding within the RACE 'COUGAR' project'. IEE Int. Broadcasting Convention, IBC 95, September 1995, vol. 413, pp. 238–243
17 Tabatabai, A.J., Jasinschi, R.S., and Naveen, T.: 'Motion estimation methods for video compression. A review', *J. Franklin Inst.*, 1998, **335B**, (8), pp. 1411–1441
18 www.celoxica.com
19 Benitez, D.: 'Performance of reconfigurable architectures for image-processing applications', *J. Syst. Archit., Euromicro J.*, 2003, **49**, (4–6), pp. 193–210
20 Horn, B.K.P., and Schunck, B.G.: 'Determining optical flow', *Artif. Intell.*, 1981, **17**, pp. 185–204
21 Cobos, P., and Monasterio, F.: 'FPGA implementation of the Horn & Shunk optical flow algorithm for motion detection in real time images'. Proc. XIII Design of Circuits and Integrated Systems Conf., Madrid, España, November 1998, pp. 616–621
22 Díaz, J., Ros, E., Mota, S., Carrillo, R., and Agís, R.: 'Real time optical flow processing system', *Lect. Notes Comput. Sci.*, 2004, **3203**, pp. 617–626
23 Fleet, D.J., and Jepson, A.D.: 'Computation of component image velocity from local phase information', *Int. J. Comput. Vis.*, 1990, **5**, (1), pp. 77–104
24 Ortigosa, E.M., Cañas, A., Ros, E., Ortigosa, P.M., Mota, S., and Díaz, J.: 'Hardware description of multi-layer perceptrons with 3 different abstraction levels', *Microprocess. Microsyst.*, 2006, **30**, (7), pp. 435–444
25 Todman, T.J., Constantinides, G.A., Wilton, S.J.E., Mencer, O., Luk, W., and Cheung, P.Y.K.: 'Reconfigurable computing: architectures and design methods', *IEE Proc., Comput. Digit. Tech.*, 2005, **152**, (2), pp. 193–207