Transactions Letters

FPGA-Based Real-Time Optical-Flow System

Javier Díaz, Eduardo Ros, Francisco Pelayo, Eva M. Ortigosa, and Sonia Mota

***Abstract*—We describe a pipelined optical-flow processing system that works as a virtual motion sensor. 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. We refer to it 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. To the best of our knowledge, this is the first study that presents a fully stand-alone optical-flow processing system to include measurements of the platform performance in terms of accuracy and speed.**

***Index Terms*—Field-programmable gate arrays (FPGAs), image motion analysis, image processing, pipeline processing, real-time systems.**

I. Introduction

THE 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 three-dimensional (3-D) reconstruction, time interpolation of image sequences, video compression, segmentation from motion, tracking, robot navigation, and time-to-collision estimation. There are several ways of recovering the 3-D information from two-dimensional (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 & Kanade's (L&K) approach [1], [2]. Several authors have emphasized the satisfactory tradeoff 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 vs accuracy tradeoff 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], [6].

We describe here a hardware implementation of the L&K algorithm. Other authors have recently described the hardware implementation of optical-flow algorithms [7]–[10], but most of them provide no results to evaluate the performance of the system, i.e., the accuracy and the computation speed. We describe a fully stand-alone working system at conventional camera frame rates of 30 Hz, with image sizes of 320×240 pixels, which to the best of our knowledge is the first description of such a system.

Manuscript received October 10, 2004; revised April 14, 2005. This work has been supported by the European Union research framework funds through the European Projects ECOVISION (IST-2001-32114) and the National Spanish Grant DPI2004-07032. This paper was recommended by Associate Editor R. Chandramouli.

J. Díaz, E. Ros, F. Pelayo, and E. M. Ortigosa are with the Computer Architecture and Technology Department, ETSI Informatica, University of Granada, E-18071 Granada, Spain (e-mail: jdiaz@atc.ugr.es; eros@atc.ugr.es; fpelayo@ugr.es; eva@atc.ugr.es).

S. Mota is with the Department of Computer Science and Numerical Analysis, University of Cordoba, Cordoba E-14071, Spain (e-mail: smota@atc.ugr.es).

Digital Object Identifier 10.1109/TCSVT.2005.861947

II. 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.

Instead of temporal finite-impulse-response (FIR), filters we have used infinite-impulse-response (IIR) filters. Temporal FIR filter requires 15 frames which are hardly affordable in embedded systems; therefore, as indicated in [11], a more efficient tactic can be adopted 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.

Another slight modification provides estimations when the aperture problem appears in the direction of the maximum gradient. We have added a small constant, α to the matrix diagonal as suggested in [12], which allows us to estimate the normal velocity field in situations where 2-D velocity cannot be extracted due to the lack of contrast information.

III. Hardware Implementation

Diverse potential applications such as robotics [13], encoding standards [14] or estimation of structure from motion [15] can benefit from the development of a customizable optical flow system of high computational power and high quality which

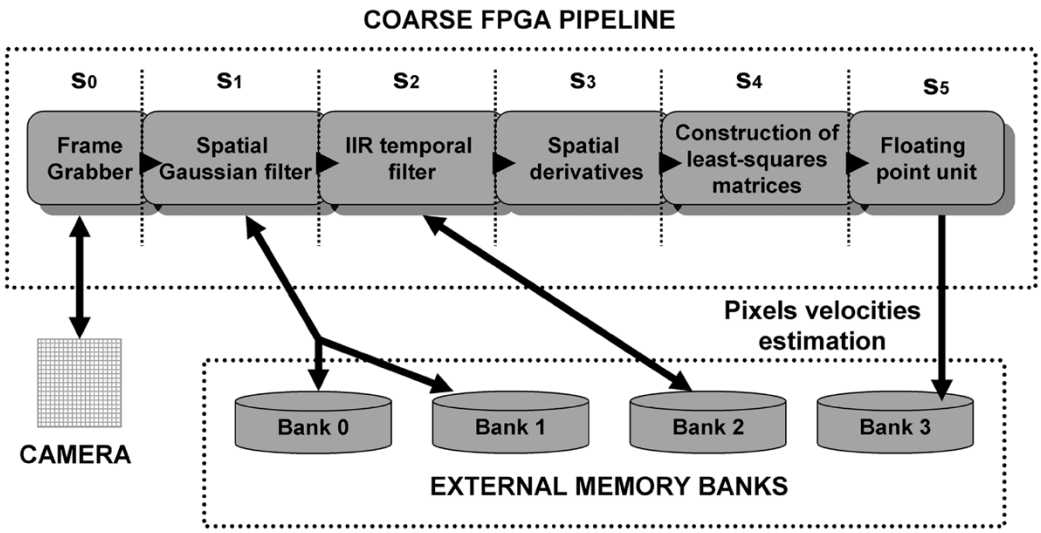

Fig. 1. Coarse pipeline processing architecture.

provides dense flow information and is suitable to be used as an embedded sensor. The customization feasibility is the key factor to address several applications with the same technology. The possibilities of working for a specific-application whereas managing design tradeoffs (such as different frame rate versus spatial resolution and customized flow accuracy versus system cost) are very advantageous. The solution we propose is based on the use of programmable logic circuits [field-programmable gate arrays (FPGAs)] where the motion computation chip can be regarded as part of a smart sensor. These circuits allow us to design a customized 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.

For our design, we have used two platforms: the first one is the RC1000-PP board from Celoxica.[1] This is a PCI bus board connected to the PC and can be used as a hardware accelerator board or as a prototype board containing a Virtex 2000E-6 Xilinx FPGA The second platform is the stand-alone RC200 board from Celoxica. This board includes camera input, video/VGA output, two 2-MB SSRAM memory banks, and an XC2V1000-4 FPGA. It is a very suitable test system for embedded applications. We have used Handel-C [16] as hardware specification language to generate the Edif input to the Xilinx ISE environment. This high-level hardware language allows us to describe the circuiuts at a register transfer level (RTL) in a very algorithmic-like way. This is relevant due to 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.

For our discussion, we will focus on the PCI board implementation. The version running on the stand-alone platform implements the same processing architecture (including new I/O controller modules). In Section IV, we will only outline the resource requirements and the improvements for the design based on the Virtex II FPGA family.

[1]Celoxica Company [Online] Available: http://www.celoxica.com

A. Hardware Development

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 a pipeline architecture, as shown in Fig. 1, the basic computational stages of which can be summarized 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 temporization 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 for integration of neighborhood velocities estimations [1].
- 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 customized 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 b in the first two stages, 12 b 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.

The basic parameters of the pipeline structure are latency (L) and the maximum number of cycles (MNCs) required during the longest stage, which is the limiting factor of the computing speed. The pipeline circuit scheme provides a computing speed

TABLE I
DETAILED SUBCIRCUIT HARDWARE REQUIREMENTS ON A VIRTEX XCV2000E. NOTE THAT THE SUM (% OF THE DEVICE IN THE FIRST COLUMN) IS LARGER THAN 100%, THIS CAN BE EXPLAINED BECAUSE THESE DATA HAVE BEEN OBTAINED BY PARTIAL COMPILATIONS AND THE SYNTHESIS TOOL MAKES A WIDE USE OF THE AVAILABLE RESOURCES. WHEN THE WHOLE DESIGN IS COMPILED IT CONSUMES 99% OF THE DEVICE

	Number of slices / (% of the device) / equivalent gates	Computing cycles / ISE maximum Clock frequency (MHz)	Memory requirements / (% of the device)
Spatial Gaussian (17 taps) S_1	220 / (1%) / 270,175	8 / 29.2	16 / (10%)
IIR filter S_2	134 / (1%) / 51,971	7 / 38.5	3 / (1%)
Spatial derivative convolution S_3	287 / (1%) / 121,296	7 / 28.0	7 / (4%)
Least square matrices construction S_4	15,288 / (79%) / 642,705	10 / 20.3	24 / (15%)
Superscalar floating point unit S_5	5,720 / (29%) / 90,993	10 / 17.4	0

(data throughput) in pixels per second (pps) that depends on the MNC and the frequency clock (f_{clk}) according to the expression $\mathrm{pps} = f_{\mathrm{clk}}/\mathrm{MNC}$.

S_4 and S_5 critical stages are the ones that most affect the tradeoff between efficiency and cost. We can modify the neighborhood area, number of convolution units, multipliers, and floating-point parallel operations. We design one-cycle floating-point hardware circuits because this works at the desired maximum clock frequency (without becoming the limiting stage) for all of 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 b of floating numbers. However, 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 synthesize more or less ways. These different alternatives lead to diverse system versions described in Section IV.

IV. HARDWARE RESOURCES CONSUMPTION STUDY

The system is designed in modules so that parallelism and bit accuracy at the different stages can be easily modified. Due to the high level of abstraction that Handel-C provides [16] it is easy to manage the parallelism of the computing circuits and the bit width at the different stages. Table I summarizes the hardware resources of the different stages using a XCV2000E Virtex FPGA for a specific implementation called HSHQ in the Table II.

The last two stages have the larger MNC values. Note that a lower MNC is possible for other stages, but there is no reason to improve them due to the other existing limiting stages. The results of the Xilinx timing analyzer are not always accurate. In fact it usually underestimates the speed at which a circuit can run: the maximum frequency allowed by the system has been experimentally measured, and it is 10–20 MHz higher than the very conservative results given by ISE. This arises because the analyzer looks at the static logic path rather than the dynamic one (cf. [17]), and, because of that, we measure experimentally the maximum working frequency. As can be seen in Table I, we have designed a system with a pipeline structure which has a global latency of 42 cycles and a maximum working frequency of 17.4 MHz evaluated by ISE (35 MHz measured experimentally).

One important aspect is that of the various possibilities for configuring the system. We have evaluated several configurations to explore different tradeoffs between accuracy, hardware cost, and computing speed. In all of these configurations, we have used the same basic architecture but with different levels of parallelism, mainly customizing stages S_4 and S_5.

Table II summarizes the main properties of the different configurations. The ones using a 5×5 average window for the least-square-matrix neighborhood are called high quality (HQ) approaches, and the ones using a 3×3 window are called medium quality (MQ) approaches. Other modifiable parameters are the smoothing and spatial derivative filter sizes. HQ and MQ approaches include 5-pixel derivative filters and 9-pixel Gaussians. A low-cost (LQ) version uses 3-pixel derivatives

TABLE II
PERFORMANCE AND HARDWARE COST OF DIFFERENT CONFIGURATIONS ON A VIRTEX 2000-E FPGA (2 MILLION GATES AND 640 Kb OF EMBEDDED MEMORY). (kpps → KILOPIXELS PER SECOND, fps → FRAMES PER SECOND). ALL OF THE PERFORMANCE VALUES WERE MEASURED USING A CLOCK FREQUENCY OF fclk = 27 MHz. THESE MEASUREMENTS (kpps AND fps) ARE UNDERESTIMATIONS BECAUSE THE COMPUTING TIME MEASURED ALSO INCLUDED DATA TRANSMISSION TO THE PROTOTYPING BOARD

Version	HSHQ	HSMQ	MSMQ	LSLQ
% device occupation	99	65	43	36
% on-chip memory	17 / 31	16 / 31	16	8
Kpps	1776	1776	625	400
Image resolution	160x120 / 320x240	160x120 / 320x240	160x120	120x90
Fps (f_{clk}=27MHz)	95 / 24	97 / 24	33	38
Max. f_{clk} (MHz)	35	35	35	35

and a Gaussian filter of the same size. If we fix the optical-flow quality of the system, another factor to take into account is the performance vs. hardware cost tradeoff. If the system works with maximum parallelism the MNC is 10. Lower cost approaches are possible if we reduce the parallelism level, thus increasing MNC. For example, we implemented a high-speed (HS) version with $\mathrm{MNC} = 10$ cycles using a three-way division unit and maximum parallelism. A slower version was implemented reducing the parallelism and thus resulting in a medium speed (MS) version. Finally, we implemented a low-speed (LS) version. Table II summarizes 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 I, the theoretical data throughput of the HSHQ is 2700 Kpps at this clock frequency. This topic is amply discussed in [18].

We also have tested the design using the standalone prototyping platform RC200 to avoid the PCI bus bottleneck. This platform includes an XC2V1000-4 FPGA with embedded multipliers. In this approach we have implemented the whole optical flow system plus Video input, VGA, and memory arbitration controller. A look-up table (LUT) for visual color representation of the velocities vector for the VGA output has also been included in the FPGA. The optical flow system implemented is a customization for this specific platform. It shares the main properties of HSHQ PCI board version but uses the embedded multipliers and several clock domains. It has a limited level of parallelism and bit-width in the floating-point unit getting a MNC value of 14 cycles.

The computing speed measured at the maximum clock frequency (40 MHz) was 2857 kpps (30 fps of 340×280 images). Now the system is faster due to the improved technology of the Virtex II and the elimination of the PCI bus. The use of a customizable approach with a high-level description language facilitates the implementation of this system on an FPGA of only one million gates (99% of resources was used). In fact, the optical flow processing algorithm only consumes 80% of the number of slices whilst the rest is occupied by the I/O controllers. The use of the embedded multipliers saves 662 slices (13% of the system resources).

V. PERFORMANCE EVALUATION

A. Comparison With Other Approaches

The implementation of the optical-flow algorithm with FPGA has only been addressed by some authors in very recent years. Using the block-matching approach, the implementation described by Niitsuma and Maruyama [7] achieves 30 fps of image size 640×480 but with high hardware cost (90% slices of a XC2V6000 FPGA) and without subpixel accuracy. Another study has been published recently [9] in which the hardware requirements are lower but no information about the system performance is provided. Based on the L&K approach, Correia and Campilho [10] 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 2303 Kpps with our PCI board system, 2857 Kpps with the stand-alone and 1666 Kpps for their approach). 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 25 fps of 160×120 pixels and this could be optimized using MMX and SSE instructions. The drawback is that it consumes all the computing resources of the machine.

B. System Performance

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 the test scheme and synthetic sequence from the comparative study made by Barron *et al.* [1], with the error measurement proposed in [19], [20]. This error measurement has been widely used in the literature and therefore it is appropriate to compare our results with previous works.

In the hardware implementation, some simplifications are made to the original model. The first row of Table III shows the accuracy of the hardware friendly L&K algorithm computed by a standard PC using double precision variables with unthresholded results. The second 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.

TABLE III
YOSEMITE SEQUENCE RESULTS USING THE ANGLE ERROR MEASUREMENT OF FLEET *et al.* [19], [20]

Model	Average Error °	Standard deviation °	Density (%)	Parameters
LK IIR software *vs.* real flow	15.91 °	11.5 °	100	λ_{min}=0, σ_{xy}=0.8, τ=2, α=1
Hardware implementation *vs.* real flow	18.30 °	15.8 °	100	λ_{min}=0, σ_{xy}=0.8, τ=2, α=1

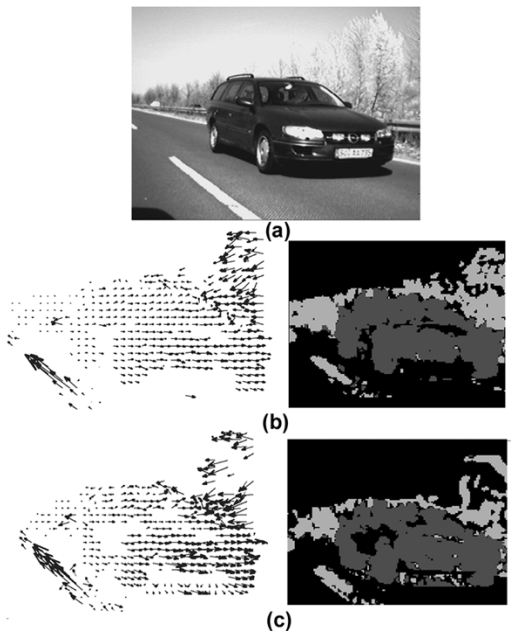

Fig. 2. Optical flow for the overtaking car. Software versus hardware estimations. (a) Original image extracted from the sequence. (b) Software result and (c) hardware result. The left-hand images use arrows to represent velocity vectors. In the right-hand images, for the sake of clarity, only leftwards (light colors) due to the landscape and rightwards (dark colors) due to the overtaking car are used to indicate the motion.

From Table III also 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 b. 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 calculated speed).

C. Real Sequences: Overtaking-Car Segmentation

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

Fig. 2 contains the image of an overtaking-car sequence seen from the rear-view mirror, together with the results of software 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, since 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 Fig. 2(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 and address further processing stages such as car tracking.

VI. CONCLUSION

The system described here shows how an optical-flow estimation circuit can be implemented using an FPGA platform as a customized DSP for a specific purpose. The paper describes a scalable architecture that can work with large image data at a conventional video-frame rate (30 fps). System performance, customization 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 modularity of the system also enables the easy alteration of the computing scheme to target different computing speed vs. hardware cost tradeoffs.

The accuracy of the estimated flow is essential for some of the possible applications outlined in the paper. 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. The results of the hardware implementation described 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 2303 Kpps with our PCI board system and 2857 Kpps with the stand-alone platform).

In the future, we plan to apply the described approach in different applications such as tracking systems, robot navigation, and video compression. 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.

REFERENCES

[1] J. L. Barron, D. J. Fleet, and S. Beauchemin, "Performance of optical-flow techniques," *Int. J. Comput. Vis.*, vol. 12, no. 1, pp. 43–77, 1994.
[2] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. DARPA Image Understanding Workshop*, Apr. 1984, pp. 121–130.
[3] H. C. Liu, T. S. Hong, M. Herman, T. Camus, and R. Chellappa, "Accuracy vs efficiency trade-offs in optical flow algorithms," *Comput. Vis. Image Understanding*, vol. 72, no. Issue 3, pp. 271–286, Dec. 1998.

[4] B. McCane, K. Novins, D. Crannitch, and B. Galvin, "On benchmarking optical flow," *Comput. Vis. Image Understanding*, vol. 84, pp. 126–143, 2001.
[5] S. Baker and I. Matthews, "Lucas-Kanade 20 years on: A unifying framework," *Int. J. Comput. Vis.*, vol. 56, no. 3, pp. 221–255, Mar. 2004.
[6] S. H. Lim and A. E. Gamal, "Optical flow estimation using high frame rate sequences," in *Proc. Int. Conf. Image Process.*, vol. 2, 2001, pp. 925–928.
[7] H. Niitsuma and T. Maruyama, "Real-Time detection of moving objects," *Lecture Notes in Computer Science, FPL 2004*, vol. 3203, pp. 1153–1157, Sep. 2004.
[8] P. Cobos and F. Monasterio, "FPGA implementation of camus correlation optical flow algorithm for real time images," in *Proc. 14th Int. Conf. Vision Interface*, 2001, pp. 32–38.
[9] S. Maya-Rueda and M. Arias-Estrada, "FPGA processor for real-time optical flow computation," *Lecture Notes in Computer Science*, vol. 2778, pp. 1103–1016, 2003.
[10] M. V. Correia and A. C. Campilho, "Real-time implementation of an optical flow algorithm," in *Proc. Int. Conf. Pattern Recognition*, 2002, pp. 247–250.
[11] D. J. Fleet and K. Langley, "Recursive filters for optical flow," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 1, pp. 61–67, Jan. 1995.
[12] E. P. Simoncelli, E. H. Adelson, and D. J. Heeger, "Probability distributions of optical flow," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, Maui, HI, Jun. 1991, pp. 310–315.
[13] H. Yamada, T. Tominaga, and M. Ichikawa, "An autonomous flying object navigated by real-time optical flow and visual target detection," in *Proc. IEEE Int. Conf. Field-Programmable Technol.*, 2003, pp. 222–227.
[14] A. J. Tabatabai, R. S. Jasinschi, and T. Naveen, "Motion estimation methods for video compression. A review," *J. Franklin Inst.*, vol. 335B, no. (8), pp. 1411–1441, 1998.
[15] T. Jebara, A. Azarbayejani, and A. Pentland, "3D structure from 2D motion," *IEEE Signal Process. Mag.*, vol. 16, no. 3, pp. 66–84, May 1999.
[16] *Handel-C Language Reference Manual, Version 3.1*, Celoxica Company, 2003.
[17] "Timing Analysis. Timing Analysis and Optimization of Handel-C Designs for Xilinx Chips," Celoxica application note AN 68 v1.1.
[18] D. Benitez, "Performance of reconfigurable architectures for image-processing applications," *J. Syst. Architecture: Euromicro J.*, vol. 49, no. 4–6, pp. 193–210, 2003.
[19] D. J. Fleet and A. D. Jepson, "Computation of component image velocity from local phase information," *Int. J. Comput. Vis.*, vol. 5, no. 1, pp. 77–104, 1990.
[20] D. J. Fleet, "Measurement of image velocity," in *Engineering and Computer Science*. Norwell, MA: Kluwer, 1992.