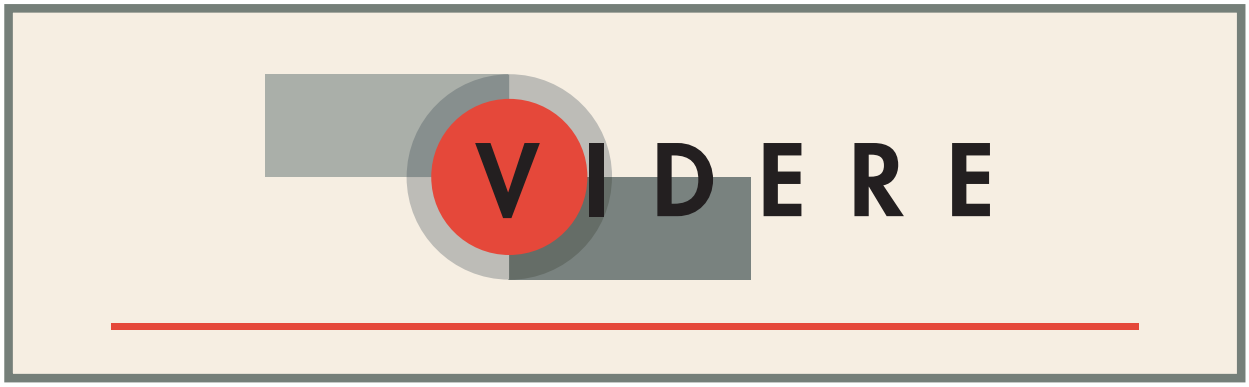


Videre: Journal of Computer Vision Research

Quarterly Journal
Winter 1998, Volume 1, Number 2
The MIT Press

Article 2

A Compact Sensor for Visual Motion Detection

Thomas Röwekamp
Liliane Peters

Videre: Journal of Computer Vision Research (ISSN 1089-2788) is a quarterly journal published electronically on the Internet by The MIT Press, Cambridge, Massachusetts, 02142. Subscriptions and address changes should be addressed to MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142; phone: (617) 253-2889; fax: (617) 577-1545; e-mail: journals-orders@mit.edu. Subscription rates are: Individuals $30.00, Institutions $125.00. Canadians add additional 7% GST. Prices subject to change without notice.

A Compact Sensor for Visual Motion Detection

Thomas Röwekamp, Liliane Peters[1]

This article presents a new sensor system for optical flow estimation. The sensor system is characterized by real-time computation based on image frames with high resolution. The core of the system is an ASIC in digital CMOS technology which performs the optical flow computation.

Although a specialized hardware is developed, the system is characterized by means of flexibility. A generic input interface makes it well suited to be connected to standard photosensitive components and the output interface can be connected to high-level image processing modules.

Keywords: optical flow, hardware, ASIC, system design, real-time, autonomous robots, gesture recognition, vergence control

1. GMD—German National Research Center for Information Technology, Schloss Birlinghoven, D-53754 Sankt Augustin, Germany. thomas.roewekamp@gmd.de, liliane.peters@gmd.de

1 Introduction

Ranging from simple animals to humans, biological systems have proved to perform amazing tasks based on visual motion analysis. To do so, during evolution special structures are developed to perform these tasks [23]. Until today, artificial vision systems were not able to solve a lot of vision tasks in real time as their biologic counterparts do. Even if they perform well on restricted subtasks, they are physically large and have high power requirements. A task that poses a major real-time constraint is the motion recognition. While in the animal eye any motion awakes the hunting instinct, the artificial motion recognition system has to support mainly collision avoidance for autonomous systems in an environment of moving (dynamic) obstacles. Until now the existing approaches cannot support autonomous platforms operating at a speed of 1 m/s.

The motion analysis task can be seen in the frame of low-level and high-level image processing. Low-level image processing tasks transform images to imagelike outputs and are characterized by high parallelism, whereas high-level image processing tasks take the low-level output and generate a symbolic answer [4]. In the case of visual motion analysis, optical flow has proved to be a useful intermediate measure and can be considered as low-level image processing [6]. The subsequent high-level modules generate a symbolic answer from the optical flow data, e.g., collision-danger alarm, 3-D structure, and motion characteristics. Due to the different requirements, low-level and high-level tasks require different computational structures. This article presents a new optical flow sensor system implemented as a low-level entry for motion analysis in real time.

In the second part of this introduction we give a short overview of the state-of-the-art development in the area of optical flow. In Section 2 we present the system architecture of the proposed optical flow sensor deducted from the constraints imposed by the chosen application range. In Section 3 we concentrate on the core of our implementation, the processing unit. The test measurements of the implemented ASIC are presented in Section 4. We are concluding our paper with some remarks in Section 5.

Previous contributions to systems for real-time optical flow computation can be divided into two groups. The first group is more-or-less *biology inspired* [19, 10, 12, 1], where the approaches of Kramer and Ancona et al are the most advanced ones. Kramer reported a sensor cell that measures the time of travel of edges between two fixed locations. The sensor cell is reported to be insensitive to stimulus contrast and selective to a high range of velocities. The approach was implemented on a 128×128 sensor array. Ancona et al presented a minaturized system based on three components: a custom CMOS photosensitive surface, a correlation circuit, and a microcontroller. In terms of size and weight,

these sensors are well suited for a mobile robot application, but the optical flow resolution is not high enough for motion segmentation. In addition, these highly specialized sensors cannot be reconfigured for multipurpose use.

The other systems, which we label *system solutions* [11, 13], deliver a motion field with high resolution at video rates. They employ specialized correlation or neural-network circuits to gain real-time performance. However, these systems are large and power consuming. Therefore, they are mainly installed on a standalone computer that receives the image information via a radio or infrared link from the mobile platform. Our brief overview of the main research approches shows that until now there is no optical flow sensor architecture that is appropriate to enhance a vision sensor for use on a mobile platform and that is adapted to the needs of low power, small size, and high speed.

2 System Architecture

To use visual motion analysis on an artificial computational system, the system has to fulfill certain requirements.

Real-time operation: predictability, throughput, and latency Usually, the task of motion analysis is not restricted to just detection of motion but aimed to trigger certain reactions on detected motion. These reactions require a system with predictable execution time and high throughput. The requested small latency makes the behavior of such a system more stable. This is a major advantage if the motion analysis is part of a control loop.

Spatial resolution The resolution of the optical flow field affects the range of computable velocities. Small velocities cannot be measured if the resolution is low. However, the number of pixels increases with the spatial resolution and consequently requires a higher number of computations per frame. In addition, high resolution enables the analysis of natural scenes with usually more than one object under motion.

Flexibility To cope with the above requirements, measurements have shown that specialized hardware is needed [16]. As special-purpose hardware has higher costs than off-the-shelf hardware, flexibility is an important factor that can increase the application area of the designed sensor and thus reduce the overall costs. Several aspects contribute to this flexibility. A miniaturized sensor system with low power consumption extends the range of application, e.g., to battery-powered mobile vehicles. A modular and scalable architecture enables the engineer to reconfigure the sensor system if constraints from applications change.

The architecture has to integrate sensorial and processing elements. Additionally, a communicational link to the sensor is required. Hence, the optical flow sensor system is composed of three basic and one auxiliary closely connected parts. (See Figure 1.)

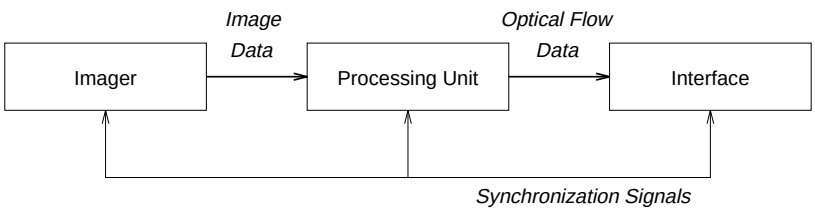


Figure 1. Overview system.

The imager The imager, or the front-end part of the system, forms the interface to the visual world. Image intensity data are acquired and transmitted to the processing unit.

The processing unit From a sequence of image data the processing unit extracts the motion information, which is represented by the components of the optical flow field.

The interface Finally, the optical flow data are forwarded to an interface (back-end), which provides the needed conversion between the low-level and the high-level image processing system.

The synchronizer Besides the image data flow, there is a need for a close synchronization of all components of the sensor system to yield maximum performance. This requirement can be easily fulfilled by using common synchronization signals that indicate valid pixels, new lines, and new frames.

2.1 Imager

The interface to the visual world is made by a cameralike component, which is named the *imager*. The purpose of the imager is to sense the visual world and to transform intensity values into electrical signals. In addition, the image data are prepared to be forwarded to the optical flow processing unit.

There exist various alternatives to implement the imager. In general it consists of two basic components: a *photosensitive surface*, and a *driving circuitry*, which adapts the chosen photosensitive surface to the connected processing unit. (See Figure 2.) A sharp distinction between these two components is not always possible. In commercial products the seeing surface is usually highly integrated and already includes some driving circuitry. Nevertheless, some additional circuitry is always necessary to couple the imager and processing unit. To build an optical flow sensor system with a high flexibility, a wide range of alternative imagers is considered. The selection is of course application driven.

For alternative implementations we considered in the first place a CMOS photodiode array [7] or a standard CCD circuit. But also other, highly specialized imagers should be considered, since they give a new opening to some interesting application areas, e.g., a space-variant

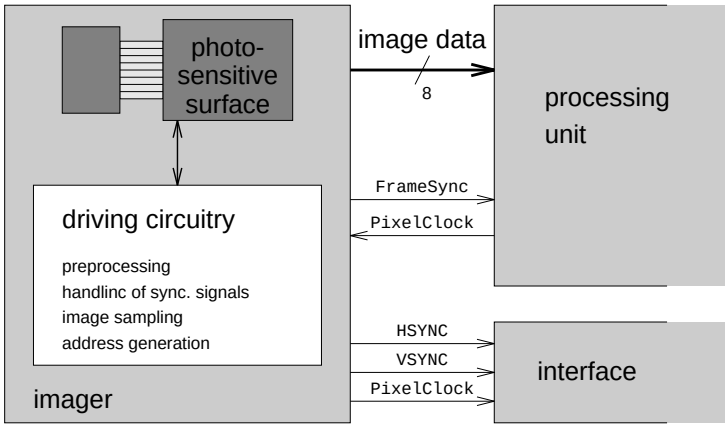


Figure 2. Imager with photosensitive surface and driving circuitry.

CMOS imager with a logpolar pixel layout [14] enables an easy extraction of optical flow attributes in an active-vision system, e.g., time-to-impact [22]. An infrared camera can be used for presegmentation of scenes under observation making just parts visible. A high-speed CCD camera enables us to analyze scenes with a high-speed motion content.

Due to the multiple possible imagers, a simple interface between the processing unit and the imager is defined. This supports the flexibility of the system. The major signals are the `PixelClock` for information request and the `FrameSync` for new frame identification. While the `FrameSync` signal indicates the start of a new frame to the processing unit, the `PixelClock` sent from the processing unit to the imager enables the sending of the image intensity data, corresponding to $xdim \times ydim$ pixels from the imager to the processing unit. The image data are coded as 8-bit data and are sent in sequential order.

The structure of the driving circuitry very much depends on the chosen photosensitive surface and may be composed by different components. It generates all driving signals for the photosensitive surface and generates synchronization signals to the processing unit and the interface. In addition it may take over tasks like AD-conversion, spatial up- and down-sampling of frames, selection of a region of interest (ROI), noise removal, error correction, and additional preprocessing. To this synchronization signals like `VSYNC` and `HSYNC` can be added, and hence the driving unit implementation takes over tasks of the synchronizer. Because of its flexibility, programmable logic is the best choice to implement the digital functions of the driving circuitry.

2.2 Processing Unit

As described in Section 2.1, the image intensity data are sent in sequential order, pixel by pixel, line by line, to the processing unit. Hence a processing architecture that takes advantage of this sequential order is best suited to compute the optical flow. A *pixel pipeline* supports such a processing scheme. However, the optical flow computation technique needs to be adapted to this processing scheme. The proposed new architecture for the processing unit of optical flow computation can be divided into three major blocks. (See Figure 3.)

Data path The data path forms the pixel pipeline. It transforms the image data into optical flow information. All data are acquired, processed, and outputted according to the pixel sequential processing scheme. The

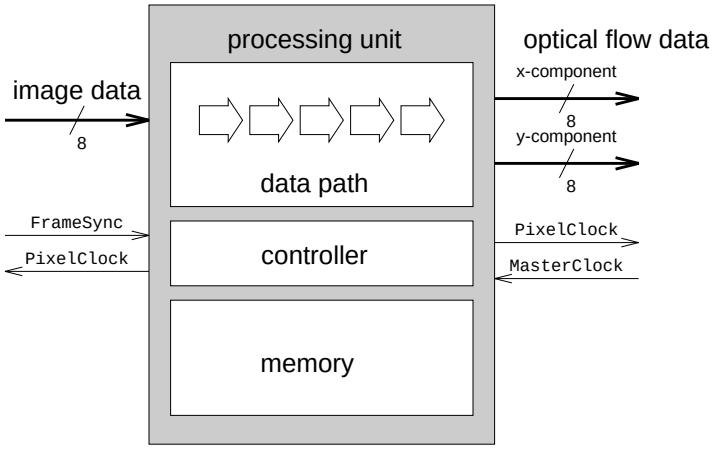


Figure 3. Block diagram of the processing unit.

pipeline is driven by the `PixelClock` signal. The output of the optical flow system corresponds to a 2-D (x, y) field of velocity vectors.

Controller The controller coordinates the timing of all elements belonging to the data path. It starts the processing upon the receipt of the `FrameSync` signal and generates the `PixelClock` to synchronize imager and interface to output. All signals are derived from an incoming `MasterClock` signal.

Memory Due to technological restrictions, the memory is not integrated into the ASIC. External components are used. Functionally they form delay lines, which are part of the data path architecture.

To assure compatibility with high-level image processing architectures, each vector component of the output is coded as an 8-bit value (byte). The format is chosen so that each output value represents motion in the range of -4 to $+4$ pixels/frame. The least significant bit (LSB) corresponds to the smallest velocity value of ≈ 0.016 pixels/frame. If we visualize the velocity data as images, the 8-bit coding gives a value of 128, corresponding to a medium gray or zero-velocity. Positive velocities appear as lighter gray, and negative velocities as darker. Figure 4 shows a sample result from a hardware simulation run. Such simulations play a central role in hardware design and allow the designer to verify the hardware before the circuit is actually going to be produced.

2.3 Interface

The computation of the optical flow from a sequence of images is only the first step in our computer vision system. A higher-level processing unit is required to transform the extracted motion information into knowledge and/or control signals. Depending on the application, this could be time-to-impact or an alarm command.

The output is via two channels (one for each component), each coded with 8-bit resolution. In addition to the data, synchronization signals need to be given. Hence digital image and signal processing systems with two input channels are well suited to interface the optical flow sensor. One possible choice is to use a digital frame grabber card in a PC to acquire the optical flow data. (See Figure 5.) Then the PC performs the higher-level tasks.

A good alternative to a frame grabber card is to interface the data directly to a DSP system. Their high I/O capabilities make them well suited to form the higher-level processing unit.

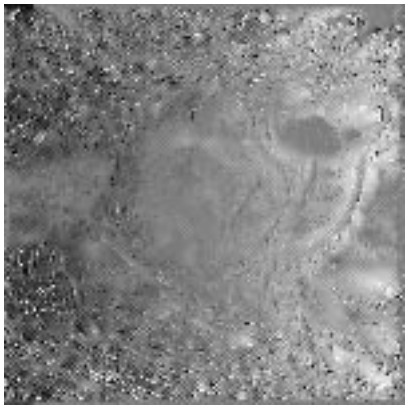
3 Processing Unit Architecture

3.1 Optical Flow Algorithm

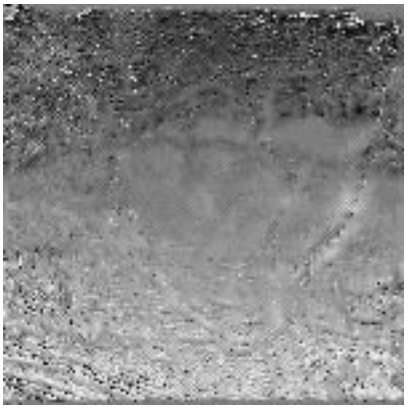
Numerous computational methods have been developed for optical flow estimation. An exhaustive survey can be read in the paper of Barron et al. [3]. One of the fundamental methods for the optical flow estimation is the technique developed by Horn and Schunck [9]. Their approach is known as a *differential technique* as it computes the optical flow from the spatiotemporal derivatives of the image intensities. As our hardware implementation started from this approach, we present it in the next paragraphs.

Let's consider the intensity I of local image regions is approximately constant under motion over a short duration of time, $dI/dt = 0$. Then the *gradient constraint equation* is derived as

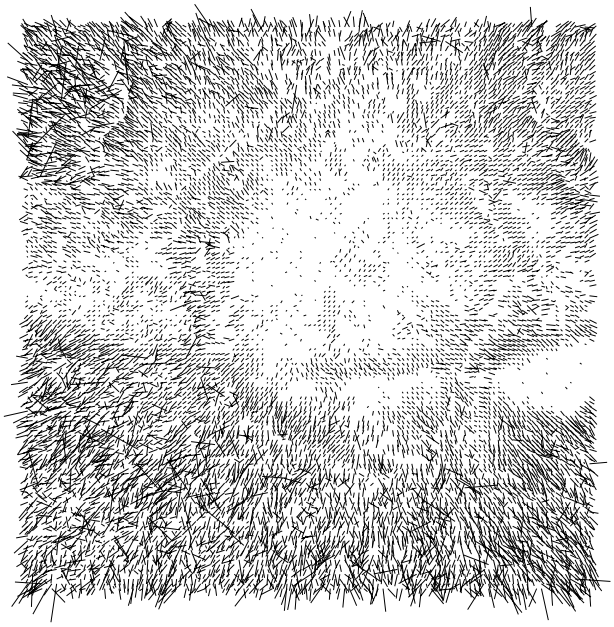
(a)



(b)



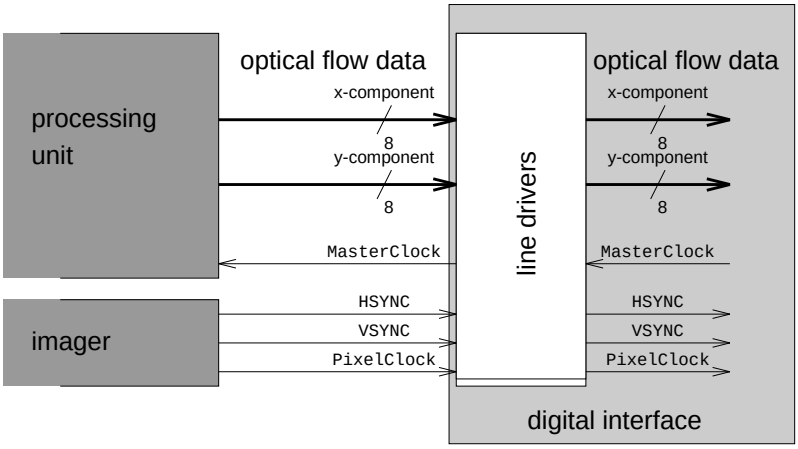
(c)



(d)

Figure 4. Frame (a) shows a sample from the diverging tree sequence provided by [2]. Among others, this sequence was used to simulate the hardware and to verify its behavior before the hardware was going to be fabricated. (b) and (c) show the x and y components of the optical flow field coded as 8-bit values and visualized as images. (d) shows a vector visualization.

Figure 5. Interface to digital frame grabber.



$$I_x u + I_y v + I_t = 0, \tag{1}$$

where u and v denote the x and y components of the optical flow vector and I_x, I_y, and I_t are the spatial and temporal derivatives of the image intensity. These derivatives can be approximated as the difference in intensity of neighboring pixels in space and respectively in time.

Equation (1) is under-determined for the computation of the optical flow components. However, as neighboring object points have similar velocities, a *smoothness regularization* term can be introduced. This allows the computation of the optical flow by minimizing a cost function derived from the gradient constraint equation and the smoothness regularization term.

This minimization problem can be solved through iteration using the relaxation formula

$$\left.\begin{pmatrix} u \\ v \end{pmatrix}\right|_{n+1} = \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} - \frac{I_x \bar{u} + I_y \bar{v} + I_t}{I_x^2 + I_y^2 + 4\alpha^2} \left.\begin{pmatrix} I_x \\ I_y \end{pmatrix}\right|_n, \tag{2}$$

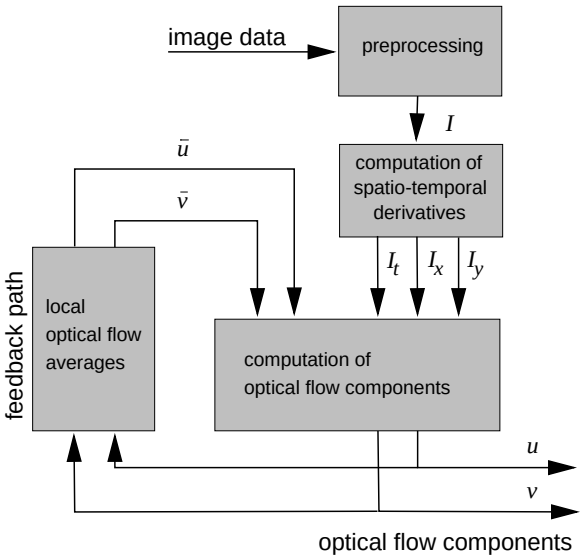
where α, known as a smoothing parameter, is a constant that weights the influence of the regularization term. The variables $\bar{u}$ and $\bar{v}$ denote the averaged local optical flow, approximated from neighboring pixels in space. Originally, n indicates the iterations to be performed for one image frame. However, assuming continuous operation and smooth object motions, this number of iterations can be reduced to one. This allows us to interpret the parameter n in Equation 2 as frame counter. At start-up, the optical flow estimation converges to its correct values after an initialization phase of a few frames.

The new proposed hardware architecture reflects the implementation of this method. Figure 6 sketches the mapping of the algorithm on the hardware architecture. It shows the partitioning of the pipeline into functional blocks: preprocessing, computation of derivatives, computation of the optical flow components, and computation of local optical flow averages.

3.2 Preprocessing

The *preprocessing block* implements a spatial Gaussian smoothing of the input image sequence. Linear filters are good for removing noise, especially Gaussian noise. A linear filter can be implemented by using the weighted sum of the pixels in the near neighborhood. The weight values are given by a convolution mask. Typically, the same type of mask

Figure 6. Data path of the processing unit.



is used for the whole image. A very simple smoothing filter, which is a rough approximation of a Gaussian smoothing, can be implemented as follows:

$$\frac{1}{8} \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{3}$$

As the operation needs a central pixel and at least the neighbors above, below, left, and right, the mask has a minimum size of 3×3. The smoothing operation is implemented through the mask operation over the whole image. This smoothing corresponds to the regularization procedures that yield solutions in ill-posed problems.

The implementation in hardware of the proposed preprocessing unit has to take into consideration some specific aspects. The data path changes the relationship among the pixels. Although the pixels in the image are spatially neighbors, in the processing unit only the left-right neighborhood relationship is kept. Due to the sequential data transmission, the above and below neighborship is lost. To compensate for this lost neighborship in the proposed hardware implementation, a shift register acting as a delay line is introduced in the preprocessing unit. A memory access scheme, that reads image data directly from RAM, can not always be implemented within the pixel sequential pipeline processing. Not all necessary memory accesses can be performed during one pixel clock cycle.

Therefore, convolutions, like the smoothing operation, are naturally not well suited to be implemented into a pixel sequential processing scheme, as they require parallel access to spatial neighboring pixels. In this case the introduction of shift registers offers a solution.

The developed *preprocessing architecture* consists of two basic building blocks: a shift register structure, and the logic unit to perform the mask operation. (See Figure 7.) The task of the shift register structure is to re-establish the spatial neighborhood of the pixels. The logic unit implements the mask proposed in this section.

The implementation challenge is to find an efficient solution for the shift register. A shift register length corresponding to two image lines is necessary to execute the mask operation. We will refer to this shift

Figure 7. Preprocessing block.

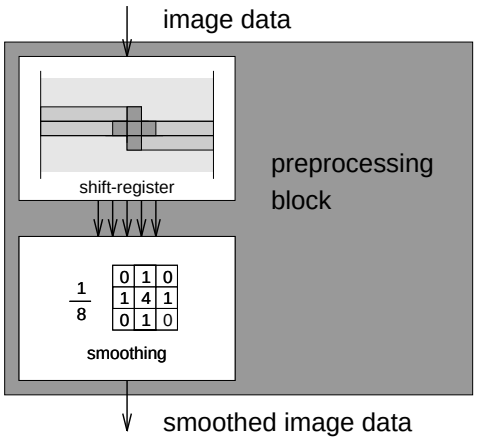
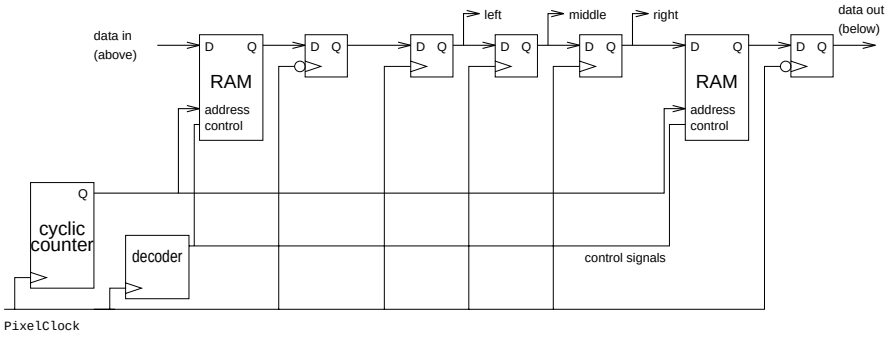


Figure 8. Schematic of the two-line shift register.



register as a *two-line shift register*. The size of the two-line shift register is dependent on the dimension of the smoothing mask.

A first approach would be to implement the two-line shift register with standard cell technology. If we consider the 0.7μm technology from Alcatel Mietec and the associated standard cell library, the result of the implementation for an image length of $xdim = 128$ is about 5mm^2. This area is too large, as we only have four two-line shift registers available if we take later processing stages into account. An implementation based on just standard cell components is therefore not possible.

Another solution is proposed by Tietze and Schenk [21]. They propose SRAMs with a successive D-flip-flop to operate as shift registers in conjunction with a cyclic counter. But this solution gives no access to intermediate data. As most ASIC foundries offer the possibility to generate optimized memories such as static SRAMs, we followed this approach in the implementation of the two-line shift register and improved it. We added a second SRAM and thus offered the possibility to access intermediate data, too. The new solution brings a decrease of at least 70% in area compared to the first one, considering the same technology. The size of a fully static, single-ported RAM of 128 words by 8-bit is $502\mu\text{m} \times 1572\mu\text{m} = 0.789\text{mm}^2$.

Figure 8 shows a simplified schematic of the two-line shift register. The main components are the two SRAM cells. The D-flip-flops cells and SRAM cells are connected by the data path, which has a width of 8 bits. By applying a cyclic address counter to the SRAM cells, the shift register functionality is achieved. The corresponding length of the shift register is determined by the cycle length of the counter. As both SRAM

cells model a shift register of the same length, a single counter is sufficient. To enable access to the central pixel and its left and right neighbor, the chain between the shift registers has to be filled with standard D-flip-flops. Each memory component labeled with *above*, *left*, *central*, *right*, and *below* provides the image intensity data, corresponding to the defined spatial neighborhood. The two-line shift register is synchronized by `PixelClock` control signal.

As mentioned in Section 2.1 some additional preprocessing steps are needed for each type of imager. To keep the system flexible, only the preprocessing steps needed for the computation of the optical flow are implemented in this unit. The image correction due to the different imager types are part of the driving unit, e.g., an offset correction needed for a CMOS photodiode array imager to remove fixed-pattern noise.

We can summarize that the preprocessing block integrated into the data path executes the following operations.

- Enhancement of the signal-to-noise ratio of the seeing surface output by smoothing.
- Improvement of the numerical differentiation of the image pixels; numerical differentiation is an ill-posed problem. Differentiation acts like a high-pass filter. In practice a smooth low-pass filter can be used to yield a well-posed problem.

3.3 Derivative Computation

The *derivative block* implements the computation of the spatial derivatives I_x and I_y and the temporal derivative I_t. (See Figure 9.) Input to the derivative block is the smoothed image data from the preprocessing block. The output of the block provides all three derivatives related to a single pixel location in parallel to the flow computation block.

Assume the image data are already smoothed; the calculation of image derivatives can then be approximated by subtracting neighboring intensity data in space and time. These operations can be seen as convolutions with simple masks:

$$[\,-1 \quad 0 \quad 1\,] \tag{4}$$

in case of the x derivative I_x, and

$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{5}$$

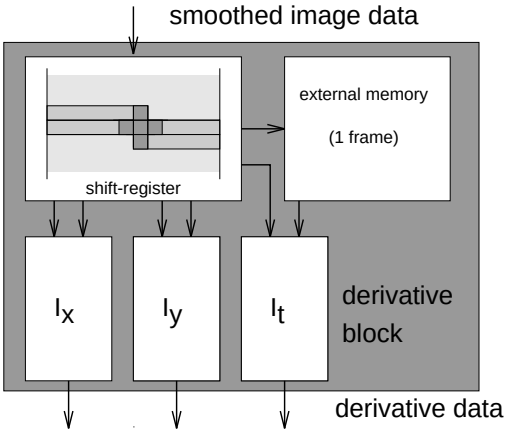


Figure 9. Derivative block.

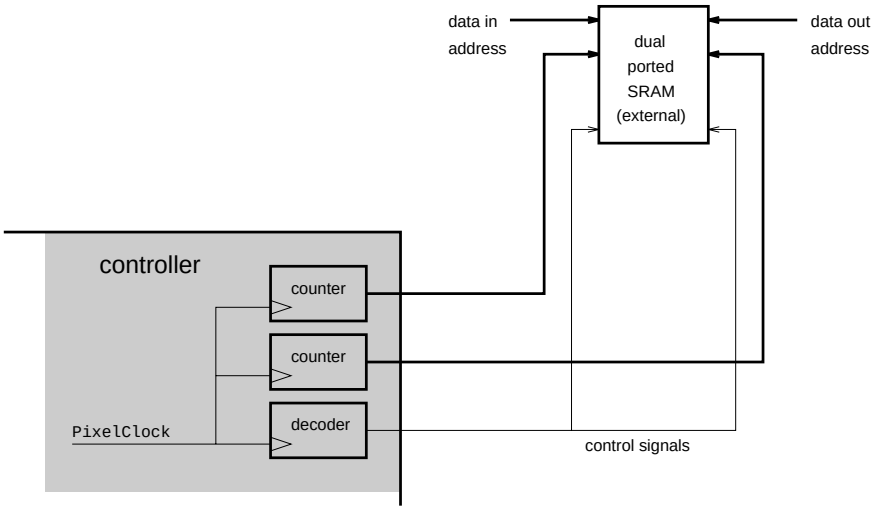


Figure 10. Implementation of the frame size shift register.

in case of the y derivative I_y. The temporal derivative I_t is the temporal difference between the current image and the previous one.

The image edges are ignored. This simplifies the circuit logic but produces wrong derivative data at image edges. The trade-off of corrupted data has been made to keep the circuitry small and fast. For the same reason there is no overflow check. (Overflows occur if either the spatial resolution or the frame rate is too low to handle the large velocities in the scene under observation.) The problem can be solved by increasing the frame rate or by selecting a larger focal length.

To some extent, the derivative block is very similar to the preprocessing block. Prior to the computations, we have the same kind of two-line shift register as introduced in the previous section. As the spatial derivatives are approximated as operations on neighboring pixels, they can be handled like mask operations. The calculation of all derivatives must be handled in parallel to maintain the pixel pipeline. However, the temporal derivative I_t needs to be handled differently. Access to data of the previous image is required.

Therefore, a second kind of shift register with a capacity corresponding to one full frame is introduced. It is named *frame shift register*. Due to its size, this shift register cannot be integrated as embedded memory on the ASIC using state-of-the-art technologies. Hence, external memories are chosen. These external memories need to form a delay line of frame length. One read/write cycle needs to be performed during one pixel clock cycle. Read and write addresses need to be generated in a cyclic manner to access input and output data corresponding to the correct pixel locations.

To satisfy these requirements, commercially available, dual ported SRAMs have been chosen. Figure 10 shows how the dual-ported SRAMs and the controller are connected. The generation of control signals and addresses is done within the controller based on the `PixelClock` signal.

3.4 Optical Flow Computation

The *flow computation block* is central to the processing unit and implements Equation (2). It is computationally the most complex block and outputs the x and y components of the optical flow, u and v. Input to the block are the spatiotemporal derivatives, I_x, I_y, and I_t, and the local averages of the optical flow, $\bar{u}(n)$ and $\bar{v}(n)$, estimated from the optical

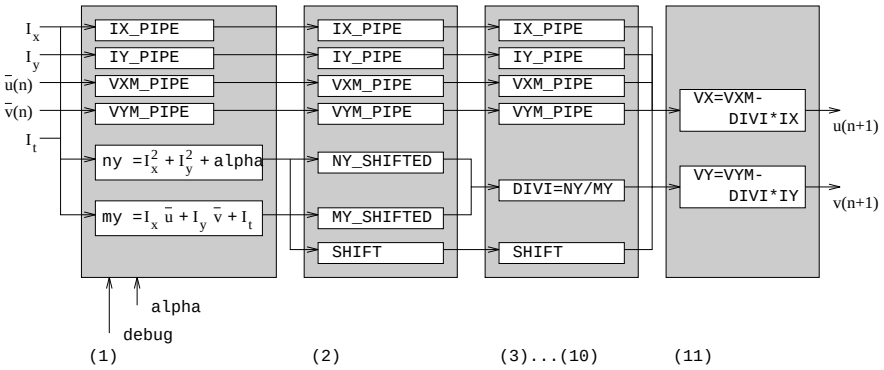


Figure 11. The flow computation block partitions the calculations in 11 pipeline steps given by the numbers in parenthesis. The related computations are indicated.

flow components of previous images. At each pixel clock cycle, all inputs must belong to the same pixel location.

The flow computation block has two additional inputs: `alpha` and `debug`. The `debug` input is a flag to output I_x and I_y instead of optical flow data. The output shows edges in the image and can be used to adjust the focus and aperture. It shows if the seeing surface and driving unit are working correctly. The signal `alpha` is a 3-bit input to the processing unit. It sets the smoothing parameter $4\alpha^2$. That way the flow computation can be adjusted to a specific application.

The calculation of the flow components cannot be done within one pixel clock cycle. An appropriate number of pipeline steps has to be chosen to find the right proportion between processing speed and silicon area. Figure 11 demonstrates how Equation (2) is partitioned in eleven pipeline steps. This is not a disadvantage, as the number of pipeline steps is small compared to the number of pixels in a frame.

In Step 1 the numerator *ny* and denumerator *my* of the division in Equation (2) are computed. The division itself cannot be performed efficiently in one pipeline step. It is prepared in Step 2. The first significant digit of *ny* and *my* is shifted to the most significant bit (MSB). The value `SHIFT` contains information to revert the shift operation later on. The core division is then performed in Step 3 to 11. It follows a compare and subtract step and delivers a value with 8 significant bits. Finally, in Step 11 we derive the optical flow x and y components. Values not used in a pipeline step are propagated to further steps by shift registers, which are indicated by `_PIPE`.

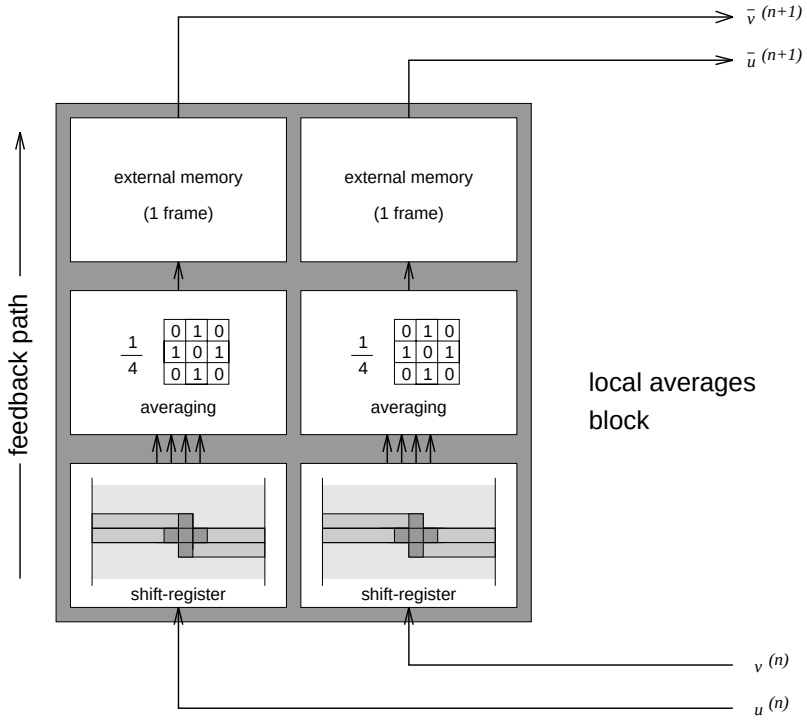
Crucial to the design of the flow computation block is the selection of data formats of intermediate values. The goal is to yield optical flow data with highest precision. Therefore, some intermediate data are extended to 16 bits. The temporal derivative is multiplied by a factor of 32, which propagates through the computations and the iterative process. This results in an LSB of the optical flow data corresponding to a velocity of 1/32 pixels/frame.

3.5 Feedback Path—Local Averages

The *local averages block* implements the feedback path. (See Figure 12.) Input data come from the optical flow output of the flow computation block. The output of the local averages block needs to be present to the input of the flow computation block. The data is expected to correspond to the correct pixel location.

The x and y components have to be handled in parallel as they are needed at the same time. So the local averages block is composed of two parallel paths. Each path consists of a two-line shift register, a

Figure 12. Local averages block—feedback path.



convolution unit, and a one-frame shift register. These components are already known from the description of the preprocessing block and the derivatives block.

The block averages the velocity components corresponding to the four neighboring pixel locations. This operation corresponds to a convolution and is implemented in a similar way as presented for the preprocessing block. The following convolution mask is used.

$$\frac{1}{4} \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{6}$$

The data corresponding to the x and y components are stored in external memories. Although separate physical devices are used as memories, the same address and control signals are applied to both components. These are generated by the controller block.

3.6 Controller

The controller is implemented as part of the ASIC and performs three tasks: generating pixel clock signals, generating enable signals, and driving external memories. Based on a `MasterClock` signal, two clocks are generated to be used inside and outside the processing unit: the pixel clock (`PixelClock`) and a 90° phase-shifted version of the pixel clock. The phase-shifted pixel clock is needed to generate the control signals for the memory components to perform one read/write cycle during one pixel clock cycle. All other operations of the processing unit are synchronously clocked with the `PixelClock` signal. The external system components, like the imager and the digital interface, are synchronized with the `PixelClock`.

Based on the PixelClock and a frame-start signal (FrameSync) all signals are generated to control the data path for optical flow computation. When the processing unit receives a FrameSync signal, it starts processing based on the newly acquired image data.

A counter is used to enable all functional units and reflects the state of processing. The counter starts counting at zero on receiving the FrameSync signal and is driven by the PixelClock. When the last valid optical flow data is generated, the counter is halted. The address counters and control signals for the external memory components are derived from the state counter.

After reading four image lines and 13 pixels, the processing pipeline is filled. The length results from the two-line shift registers and the number of computational steps. First optical flow data are then given to the output. This delay defines the latency of the optical flow processing of the system. If the frame rate is 50 frames/s and the frame size is 128×128 then the latency is about 0.63 msec. A complete optical flow field is computed when a complete image is acquired and the pipeline is filled. Here the pipeline-processing scheme shows advantages in terms of latency compared to other architectures. This makes it well suited for control loops which will show greater stability.

In general, a new image can be acquired before the processing of the preceeding image is finished. To do so, a second state counter is introduced. That way, successive images can be handled using both counters alternatingly.

4 Implementation Results

To test the architecture a frame size of 128×128 was chosen. The design was mapped on the Alcatel Mietec 0.7μm CMOS process (single poly, double metal) and was successfully fabricated. Using the standard cell library from Alcatel Mietec, all logic is implemented by automatic synthesis. Nevertheless, shift registers are much too large to be implemented in standard cells. Therefore, we chose to implement the shift registers with SRAMs, which are provided by the foundry as compiled cells, in conjunction with cyclic address counters. This approach made it possible to implement all shift registers on chip.

The ASIC—the ROFlow-Chip—has an area of $6.8 \times 6.9\text{mm}^2$ including approximately 15k gates, 1KB of memory, and 150 pad cells. Figure 13 shows a photograph of the ASIC with the location of the functional blocks. First functional tests of the complete sensor system—the ROFlow-Sensor—have shown that the fabricated ASIC performs according to simulations and synthesis constraints. An example output of the optical flow chip is presented in Figure 14. Image velocity is mainly detected along the contour of the object under observation. It can be seen that a person is walking from the left to the right. While the ASIC is able to process up to 1500 frames/s, the seeing surface currently limits the operation speed to 50 frames/s.

Whereas real-life scenes give only qualitative ideas about the functionality of the sensor system, some quantitative data are derived from experiments with translational motion parallel to a textured wall (Figure 15) or orthogonal towards an obstacle (Figure 16). These experiments evaluate the optical flow data and compute characteristics of the flow field.

In the first experiment the sensor moves parallel to a textured wall. The photosensitive surface points towards the wall. The sensor moves

Figure 13. Photograph of ROFlow-Chip.

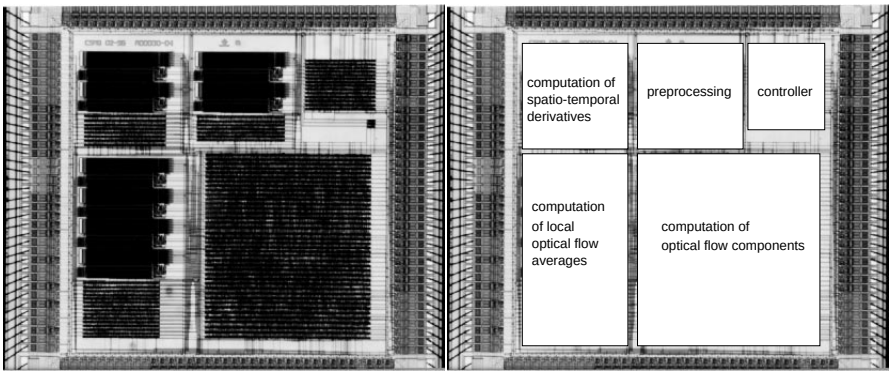
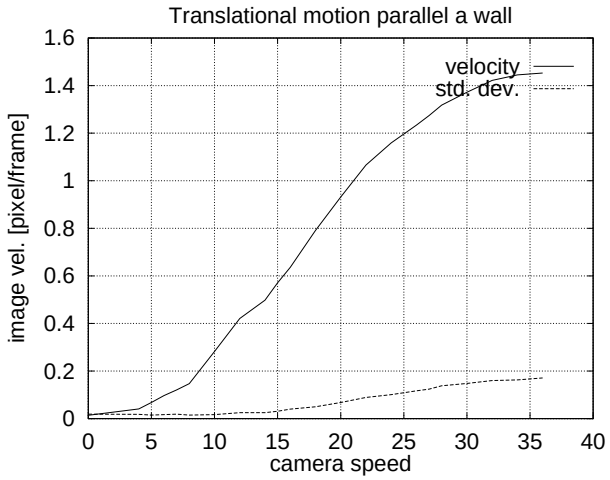


Figure 14. Somebody walking in front of the sensor system from the left to the right.

(a) (b) (c)

Figure 15. Data measured with the sensor translating parallel to a textured wall. The experiment is done with 37.6 frames/s. The values of the x axis are proportional to the camera's speed.



with various velocities proportional to the values at the x axis of the diagram in Figure 15. The frame rate is 37.6 frames/s. The diagram illustrates the range of velocities that can be detected. It gives the x component of the optical flow averaged over all pixels along with its standard deviation; the y component is 0. The averaged image velocities are proportional to the sensor's speed over approximately one decade. The measured standard deviation increases with increasing speed. However, the results depend to some degree on the kind of wall texture and the setting of the smoothing parameter $4\alpha^2$. The actual range of detectable velocities can be adapted to the observed scene by the selection of an appropriate frame rate. A higher frame rate is required to detect smaller velocities, a lower for larger velocities.

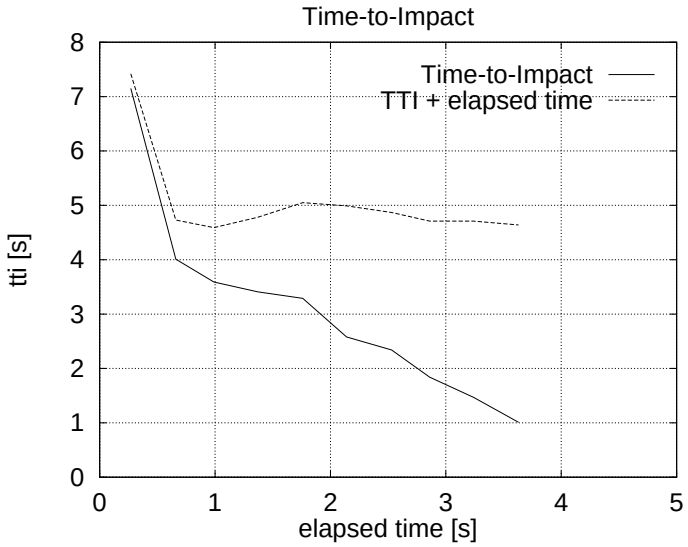


Figure 16. Data measured in a time-to-impact experiment. The curves show that elapsed run-time and the estimated time-to-impact add to a nearly constant value after a short initialization phase. The frame rate is 18.8 frames/s.

The second experiment is dedicated to characteristics, which can be derived from the flow field and can be used, for example, to control a vehicle. Here the first-order derivatives are used to compute the time-to-impact (TTI), which is defined as the time needed to hit a surface if velocities do not change [5]. A vehicle is driving orthogonal towards an obstacle with constant velocity. From the sensor mounted on the vehicle the TTI is estimated. If this value falls below a critical value, the vehicle is stopped. First, the optical flow data are computed at a rate of 18.8 frames/s. Second, a PC is used to evaluate the flow data following a least-squares approach to derive the TTI values. Figure 16 shows results from such an experiment. It gives the estimated TTI values in seconds corresponding to the elapsed time of the experiment. If the TTI estimation is correct, then the TTI and the elapsed time add to a constant value, which corresponds to the overall duration of the experiment, from start to impact. In the documented experiment the impact would happen at about 4.6 seconds after start.

5 Concluding Remarks

This article presented a new sensor system for real-time optical flow estimation. Specifically, it is shown how an existing algorithm is implemented to a real-time hardware under given constraints. A description of the architecture and results from a prototypical implementation are given.

Following are some additional remarks concerning the performance, applications, and future work.

Performance The ROFlow-Sensor—a prototype implementation—computes a flow field of 128×128 vectors at a rate of 50 frames/s. More-advanced systems can be built as the architecture is scalable and can be reconfigured with various components as front-end and back-end. Currently the frame rate is limited due to the selected imager. Using a high-speed imaging device, the full frame rate of 1500 frames/s can be exploited. The ROFlow-Chip is designed using a hardware description language. By setting parameters for the number of rows and columns,

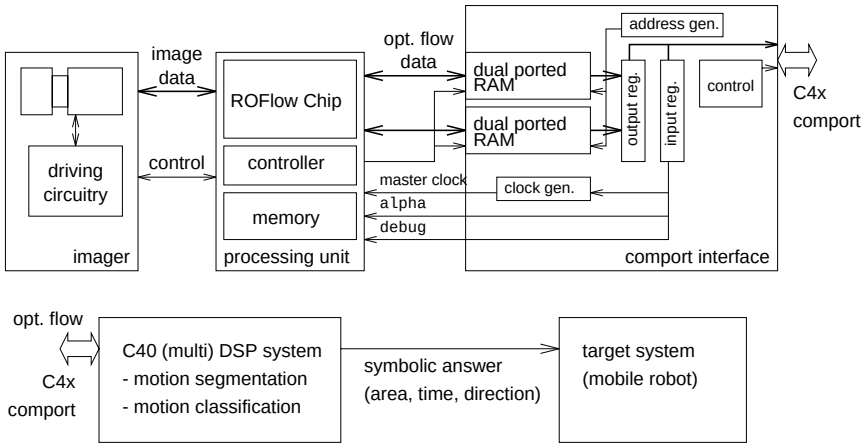


Figure 17. Collision avoidance system. The comport interface is the communicational link between sensor and DSP system.

a circuit can be implemented to process larger frame sizes. However, because of the pipeline processing, the maximum frame rate decreases. But with resolutions of 512×512 pixels, a frame rate of 90 frames/s can still be achieved, suppose we use the same CMOS technology to design the hardware.

Applications To illustrate the advantages of a modular system design and to show the flexibility in use of the ROFlow-Chip, we briefly address architectural concepts towards two different application areas. The case studies in the following are not meant to evaluate the applications, but to demonstrate how the ROFlow-Sensor can be configured with various front- and back-end components to serve the specific needs.

The first application study is dedicated to autonomous mobile vehicles and demonstrates the use of a specialized interface. In this area, the use of visual motion analysis is already proved. (See [8, 17, 18].) Collision avoidance is the most essential task and is prerequisite for useful tasks, such as mail delivery in environments populated with humans. The main goal for optical flow computation in hardware is to operate a vehicle at higher speeds. For the collision-avoidance task, we propose a system based on the ROFlow-Sensor for optical flow computation and a DSP system for motion segmentation and classification. The answer of the collision-avoidance system is information about area, time, and direction of a possible impact. To communicate between ROFlow-Sensor and DSP system, a digital interface is required. Figure 17 shows an approach of such interface for the C4x communication port [20]. On one hand it allows control of the sensor's frame rate and smoothing parameter $4\alpha^2$. On the other hand it transfers the optical flow data to the DSP system.

The second application study addresses man-machine-interfaces and illustrates the use of the ROFlow-Chip in combination with a specialized camera. The interpretation of hand signs plays a central role in this area [15]. Quantitative motion analysis can help to interpret such signs as they are usually closely related to motions. Such systems can be used in medicine to allow a surgeon to give commands to an assisting system. Reliability and real-time reaction is required. Relevant motion in the foreground has to be segmented from any background motion. This is computationally expensive and can be simplified by the use of proper illumination in the area of interest. To avoid any conflict

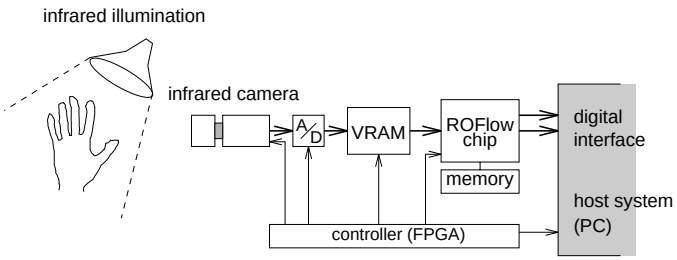


Figure 18. Hand sign experiment. Infrared-sensitive camera with video signal output as front-end.

with other sources of illumination, an infrared light source in combination with a specialized infrared-sensitive camera is chosen. Figure 18 shows a system approach, in which the ROFlow-Chip is connected to the infrared-sensitive camera front-end. AD conversion, buffering, and control is introduced to form the interface in between. This way, a field of 128×128 image data can be given to the optical flow computation.

Future The low-level vision task of optical flow computation in real time is the first but fundamental step towards real-time motion analysis applications. The next step needs to focus on subsequent higher-level tasks, which extract a symbolic answer from the optical flow output of the sensor system. Also in this case, application areas need to be analyzed to develop suitable architectures to cope with the real-time constraint. In the future, the combination of real-time, optical flow computation as low-level task and the subsequent higher-level, motion analysis task will provide artificial computational systems with abilities known from biological examples.

6 References

[1] Ancona, N., Creanza, G., Fiore, D., Tangorra, R., Dierickx, B., and Meynants, G. A real-time, miniaturized optical sensor for motion estimation and time-to-crash detection. In *Proc. of AEFPAC/EUROPTO*, pages 675–681, Berlin, SPIE, 1996.

[2] Barron, J., Fleet, D., and Beauchemin, S. Performance of optical flow techniques. Technical Report TR No. 299, Dept. Computer Science, Univ. of Western Ontario, London, Ontario, Canada N6A 5B7, 1993.

[3] Barron, J., Fleet, D., and Beauchemin, S. Performance of optical flow techniques. *Int. J. of Computer Vision*, 12(1):43–77, 1994.

[4] Choudhary, A. and Ranka, S. Parallel processing for computer vision and image understanding. *IEEE Computer*, 25:7–10, 1992.

[5] Cipolla, R. and Blake, A. Surface orientation and time to contact from image divergence and deformation. In G. Sandini, editor, *Computer Vision—ECCV'92*, pages 187–202, Springer Verlag, 1992.

[6] DeMicheli, E., Torre, V., and Uras, S. The accuracy of the computation of optical flow and the recovery of motion parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):434–447, 1993.

[7] Dierickx, B., Scheffer, D., Meynants, G., Ogiers, W., and Vlummens, J. Random addressable active pixel image sensors. In *Proc. of AEFPAC/EUROPTO*, pages 2–7. Berlin, SPIE, 1996.

[8] Fernmüller, C. Passive navigation as a pattern recognition problem. *Int. J. of Computer Vision*, 14(2):147–158, 1995.

[9] Horn, B. K. P. and Schunck, B. G. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[10] Hutchinson, J., Koch, C., Luo, J., and Mead, C. Computing motion using analog and binary resistive networks. *IEEE Computer*, 21(3):52–63, 1988.

[11] Inoue, H., Inaba, M., Mori, T., and Tachikawa, T. Real-time robot vision system based on correlation technology. Technical Report, Department of Mechano-Informatics, The University of Tokyo. Annual Report, 1993.

[12] Kramer, J. Compact integrated motion sensor with three-pixel interaction. *IEEE Trans. on PAMI*, 18(4), 1996.

[13] Lee, J. and Fang, W. VLSI neuroprocessors for video motion analysis. *IEEE Trans. on Neural Networks*, 4(2):178–191, 1993.

[14] Pardo, F., Boluda, J., Perez, J., Felici, S., Dierickx, B., and Scheffer, D. Response properties of a foveated space-variant CMOS image sensor. In *Proc. of ISCAS*, volume 1, pages 373–376, 1996.

[15] Pavlovic, I., Sharma, R., and Huang, T. S. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Trans. on PAMI*, 19(7), 1997.

[16] Röwekamp, T., Platzner, M., and Peters, L. Specialized architectures for optical flow computation: A performance comparison of ASIC, DSP, and multi-DSP. In *Proceedings of the 8th ICSPAT*, volume 1, pages 829–833, San Diego, California, USA, 1997.

[17] Santos-Victor, J. and Sandini, G. Visual behaviors for docking. Technical Report TR 2/94, LIRA-Lab, DIST, University of Genova, 1994.

[18] Santos-Victor, J. and Sandini, G. Uncalibrated obstacle detection using normal flow. *Int. J. Machine Vision and Applications*, 9(3):130–137, 1996.

[19] Tanner, J. and Mead, C. An integrated optical motion sensor. In S.-Y. Kuns, R. E. Owen, and J. G. Nash, editors, *VLSI Signal Processing II*, pages 59–76. IEEE Press, New York, 1986.

[20] Texas Instruments. *TMS320C4x User's Guide*. TI Inc., 1996.

[21] Tietze, U. and Schenk, C. *Halbleiter-Schaltungstechnik*. Springer Verlag, 1989.

[22] Tistarelli, M. and Sandini, G. On the advantages of polar and log-polar mapping for direct estimation of time-to-impact for optical flow. *IEEE Trans. on PAMI*, 15(4), 1993.

[23] Ullman, S. Analysis of visual motion by biological and computer systems. *IEEE Computer*, 14:57–69, 1981.