Real-Time Implementation of an Optical Flow Algorithm

Miguel V. Correia,* Aurélio C. Campilho
Instituto de Engenharia Biomédica,
Faculdade de Engenharia da Universidade do Porto,
Dept. Eng. Electrotécnica e de Computadores,
R. Dr. Roberto Frias, 4200-465 Porto, Portugal.
{mcorreia,campilho}@fe.up.pt

Abstract

This paper presents the implementation of an optical flow algorithm on a pipeline image processor. The overall optical flow computation method is presented and evaluated on a common set of image sequences. Results are compared to other implementations according to two different error measures. Due to its deterministic architecture, this implementation achieves very low computation delays that allow it to operate at standard video frame-rate and resolutions. It compares favorably to recent implementations in parallel hardware.

1. Introduction

Optical flow algorithms generally demand for high computational power and huge storage capacities. On the other hand, the many implementations of optical flow algorithms, reported by several authors are mostly experimental ones with no intent to apply them in any practical situation requiring outputs at standard video frame-rate.

Two recent implementations report processing times in the order of tens of seconds per frame [6, 4] for image sequences of moderate spatial resolution. Recently, we have applied optical flow computation as a tool to help characterize visual motion in studies of the human visual perception [2, 9]. It was found that, in order to be useful, optical flow must be computed with short processing time spans.

The objectives of the work presented in this paper were to develop an implementation of optical flow computation that provides a dense characterization of visual motion from image sequences at video frame-rates.

Section 2 discusses the issue of optical flow computation. In section 3 we propose and present the method and its implementation for optical flow computation in real-time on a pipeline image processor. Results are presented and discussed in section 4.

2. Computation of Optical Flow

Optical flow consists on the time variations of brightness patterns produced in an image. On living beings with visual ability, this continuously changing image appears in the retina while in artificial systems, it is captured by a light sensor in a camera.

The study of Barron et al. [1] on the performance of methods for optical flow computation shows several results of correlation, gradient, energy and phase-based methods on different kinds of image sequences. Also, according to the work of Simoncelli [10], determining optical flow can be performed through the implementation of spatio-temporal differential filters. Simoncelli even argues that correlation techniques can be transformed into these filtering operations and that the other methods, including energy or phase-based, can also be considered different formulations resulting in special cases of spatio-temporal filtering. Therefore, a careful evaluation of different methods led us to adopt the method of Lucas [7] to compute optical flow by the following reasons: low computational complexity and rather good accuracy, according to the comparison of Barron et al. [1].

In this gradient-based method, velocity \( \mathbf{v} (x, t) \) is computed from first-order derivatives of image brightness, using the well known motion constraint equation:

\[
\nabla I(x, t) \cdot \mathbf{v}(x, t) + I_t(x, t) = 0
\]  

(1)

where \( \nabla I(x, t) \) denotes the spatial gradient of image intensity, \( I(x, t) \), and \( I_t(x, t) \) is its partial temporal derivative. In order to obtain the two components of the velocity vector we impose a constant parametric model on the velocity field and assume spatial and temporal coherence locally, as in [7].
The optical flow on each pixel \( \mathbf{v}(x, t) \) is obtained by computing:

\[
\mathbf{v}(x, t) = [A^T W A]^{-1} A^T W \mathbf{b} \tag{2}
\]

where \([A^T W A]\) is a \(2 \times 2\) matrix and \(A^T W \mathbf{b}\) is a \(2 \times 1\) matrix given, respectively, by:

\[
A^T W A = \begin{bmatrix}
\sum w I_x^2 & \sum w I_x I_y \\
\sum w I_x I_y & \sum w I_y^2
\end{bmatrix} \tag{3}
\]

\[
A^T W \mathbf{b} = \begin{bmatrix}
\sum w I_x I_t \\
\sum w I_y I_t
\end{bmatrix} \tag{4}
\]

where summation is taken over a spatial neighborhood \(\Omega\) with the weighting factor \(w\) performed by additional smoothing operations.

For the implementation of the partial differential filters, we consider a separable smoothing filter in space-time, so that the spatial and temporal components can be computed independently. To compute spatial derivatives we use a FIR smoothing filter with a support of \(5 \times 5\) pixels followed by numerical differentiation with a \(3\) tap FIR filter, both proposed by Simoncelli [10]. The temporal filter was designed according to the work of Fleet and Langley [3] by a cascade of two truncated IIR exponentials. Temporal smoothing and differentiation are both obtained simultaneously with this filter (see [3] for proof), improving computational cost. Total temporal delay is in the order of \(3\) frames, largely reducing storage requirements.

3. Real-Time Implementation

The real-time implementation was developed for the MaxVideo200 pipeline image processor with resort to VEIL–Virginia’s Extensible Imaging Library 1. The C++ class hierarchy of VEIL creates a high level abstract model following an algorithmic approach based on a graph of operators [8]. VEIL schedules the graph so there are no conflicts in the use of the available hardware resources.

The overall smoothing and differentiation operations, as well as the integration over a local neighbourhood were implemented for the pipeline image processor as shown in figure 1. Each block represents a VEIL operator that maps to the MaxVideo hardware. Spatial smoothing is performed by Convolve operators; the Smear operator performs temporal smoothing and differentiation; while the Look-Up Table operators implement the square and product with an output resolution of \(16\) bits. The two components of velocity result from simple arithmetic implemented as shown on the diagram of figure 2. Due to hardware constraints, the arithmetic operations are limited to \(16\) bit integer resolution and, since there is only one arithmetic unit local memory has to be used for temporary storage indicated by the grey blocks. The Maximum operator and the Look-Up Tables provide numerical scaling in order to obtain the most significant digits of the result in a fixed-point \(16\) bit resolution.

4. Results

This implementation was tested on a common set of synthetic and real image sequences, illustrated in figures 3 and 4. The vector fields in figures 5 and 6 represent the computed optical flow.

The error measures used for comparison were the angular error relative to the correct image velocities in the synthetic image sequences, as defined in [1], and the image reconstruction root mean square error obtained by bicubic spline (BS) and linear displacement (LD) interpolations, as defined by [5], for both kinds of images.

It can be seen, in table 1, that the angular error is substantially worse in the real-time implementation. This result is mainly due to the smaller filter kernels and the limited integer resolution used in the real-time implementation.

When comparing image reconstruction errors we observe, from tables 2 and 3, that the real-time implementa-
The diagram performs worse in the pure translation case, but it approaches the performance of Barron et al. implementation on all other cases. This can be justified by the fact that pure translation adheres more closely to the assumption of local constant velocity and benefits from the larger filter kernels of the Barron et al. implementation. Densities are also very similar in both implementations. The limited integer resolution of the real-time implementation completely justifies the other small differences in performance errors.

On the other hand, the processing time for the Yosemite sequence, with a spatial resolution of $252 \times 316$ pixels, is 47.8 ms per frame for the real-time implementation. The processing time reported by Fleury et al. [4] on a 4 processor parallel system is of 10.7 s. Therefore, our implementation presents a considerable gain in processing speed with a small loss in accuracy. Standard video frame rate can be achieved by using a spatial resolution of 100000 pixels or by using additional arithmetic units on the pipeline image processor.

5. Conclusions and Future Work

We described an implementation of an optical flow algorithm on a pipeline image processor. The method is gradient-based and assumes a constant parametric model of velocity fields with spatial and temporal coherence. The deterministic architecture of this implementation achieves very low computation delay, exhibiting a performance superior to parallel implementations reported by others. The operation at video frame rate is also achieved, avoiding the need to store large quantities of image data.

The next step will be to segment optical flow in order to determine egomotion and motion of other objects in the scene. This will allow to obtain a more compact and reliable characterization of visual motion.

The final goal is to provide a real-time, practical tool for
the computation and analysis of optical flow in studies of visual motion perception. It may also be applicable to autonomous navigation at the early stages of motion sensing.

References


