Cellular Automata based Optical Flow Computation for “Just-in-Time” Applications

Giovanni Adorni, Stefano Cagnoni and Monica Mordonini
Department of Computer Engineering - University of Parma
Parco area delle Scienze, 181A - 43100 Parma, Italy
{adorni,cagnoni,mordonini}@CE.UniPR.IT

Abstract

Real-world tasks often require real-time performances. However, in many practical cases, “just in time” responses are sufficient. This means that a system should be efficient enough to operate on-line and to be usable in reactive systems, while being robust enough for the specific task they are performing.

This paper illustrates a new “just-in-time” technique for feature-based optical flow computation on a cellular automata paradigm and, as a case study, its implementation on a special-purpose architecture for cellular automata. Feature extraction is performed by means of a simple geometrical coding based on the local morphology of edges, which allows its description in terms of the cellular automata paradigm and reduces its temporal complexity.

The experimental results demonstrated that the algorithm performs well both in controlled and uncontrolled environments.

1 Introduction

Motion detection and interpretation are important issues for event perception in artificial vision systems dealing with complex and dynamic scenes. Since the early years of computer vision, optical flow computation has received significant attention from researchers, and several methods have been proposed [4]. Basically such methods can be classified into two main groups: (i) feature-based approaches, (ii) gradient-based approaches. In feature-based approaches, two steps are usually performed: feature extraction and matching. In the feature extraction step some features, such as image edges, corners, texels and other structures that are well localized in two dimensions, are detected in the image.

In the feature matching step, such features are tracked from frame to frame. The second step requires solving the so-called “correspondence” problem, in which the same features detected in the earlier phase must be identified in a subsequent scene and their displacements which have occurred in the meanwhile must be evaluated (see, for example, [3, 15, 12]).

In the gradient-based approach, first proposed in [9], the algorithms use spatial and temporal partial derivatives to estimate image flow for every point of the image: knowledge of image motion must be introduced in the form of constraints to obtain algorithms that are not too computationally expensive. Other more recent gradient-based algorithms, whose approaches differ from the original work by Horn and Schunk, are discussed, for example, in [11, 5].

In this paper we describe an algorithm for feature-based optical flow detection, suitable for robotics applications. Generally, the two major goals to be achieved in robotic applications to real tasks are robustness of real-world operation and limitation of computational complexity [8]. However, in many practical cases, real-time performances are not strictly necessary, “just in time” responses being sufficient [2]. Algorithms should be efficient enough to operate on-line and to be usable in reactive systems, detecting motion accurately enough for specific tasks [1]. To limit the computation temporal complexity of motion perception many techniques exploit the massively parallel nature of image processing (see, for example [6, 7]). The proposed algorithm is suitable for just in time applications and can be employed both in controlled and uncontrolled environments. Feature extraction is performed by means of a simple geometrical coding based on the local morphology of the edges, which allows its description in terms of the cellular automata (CA) paradigm [13] and reduces its temporal complexity.

2 The CA-based approach

Uniform cellular automata are discrete dynamical system whose behaviour is completely specified in terms of a single local relation. A cellular automaton can be thought of as a stylized universe in which space is represented by a uniform grid of cells, each containing a few bits of data; time advances in discrete steps and a single rule, that can be encoded in a look-up table, is applied, at each time step, to compute the new state for each cell from that of its close neighbors. Thus, the system's laws are local and uniform.

The aim of our research is to build a tool that can be included in the vision system of autonomous robots operating in both indoor and outdoor environments. For real-world applications, an optical flow computation algorithm must find a good trade-off between the requirements of real-time operation and good reliability. To reduce temporal complexity, image processing algorithms can exploit their intrinsically parallel nature, if they are designed according to a suitable parallel computation model. The proposed algorithm relies on the cellular automata paradigm, which is particularly efficient for the implementation of local uniform operations, such as those that are usually performed in most image processing applications.

An ideal optical flow algorithm suitable for CA implementation should be characterized by the following properties:

1. it should work on integers;
2. computation should be based on binary operators;
3. image processing should be performed by means of morphologic operators;
4. if image pre-processing is required, it should satisfy the above-mentioned requirements.

To reach the goal of low computational complexity, we chose to analyze only two subsequent frames of the video sequence received from the camera: the frame acquired at time t (*New-image* in the following) and the frame acquired at time $t - \Delta t$ (*Old-image* in the following).

The high-level architecture of the proposed feature-based algorithm is sketched in Figure 1.

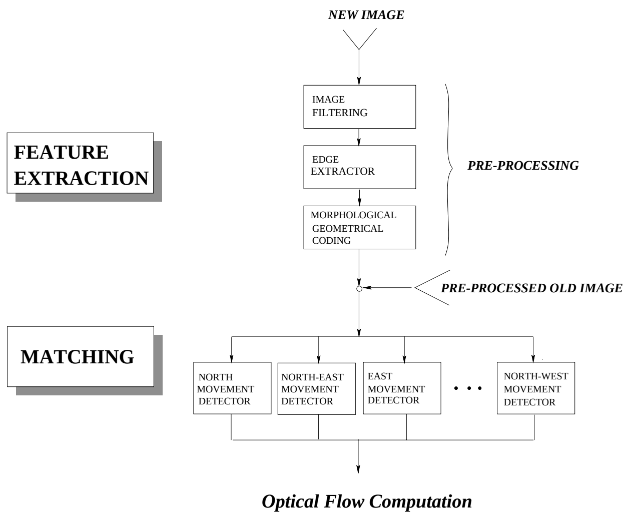

Figure 1. The architecture of the algorithm.

2.1 Feature extraction

In the feature extraction phase, images are first pre-processed to reduce noise by removing the three least significant bits from each pixel. This choice is derived from a statistical analysis of the camera CCD random noise. After pre-processing, Robert's edge extractor operator [14] is applied, producing a binary image in which pixels that belong to edges are black (0) while all other parts of the image are white (1). The choice of this edge extractor was made after a comparison between different methods applied to the pre-processed image and was based on practical considerations to keep complexity low.

Using such low-depth images, it is possible to achieve a very efficient implementation of the following steps. However, even after this step, a direct pixel-to-pixel matching is unreliable. Furthermore, the detected edges may be affected by deformations due to missing points (holes) and extra points. To deal with these problems the algorithm groups similar edge configurations into classes that also include possible noisy instances of a basic edge shape. A simple geometrical encoding based on the local morphology of the edges is used, compressing the geometrical properties of a 3×3 pixel window into a 4-bit integer. A statistical analysis of the population of the 3×3 pixel edge shapes in a long sequence of frames has been used to identify the most frequently occurring shapes. The experiments have demonstrated that the frequency with which different shapes appear in the sequence depends on the kind of scene (indoor, outdoor), on image resolution and on the edge extractor used. After examining several indoor and outdoor frame sequences, 16 shapes of size 3×3 pixels were identified which are probably the most common and may serve as a fundamental set of edge shapes. The final shape set is showed in Figure 2, in which on the bottom of each model the corresponding 4-bit integer code is reported.

Therefore, to encode the binarized frame the following procedure is adopted: for each pixel, the 3×3 window (centered in it) is examined and the bit representing the pixel value is replaced by the 4-bit code of the corresponding model. When it is impossible to find an exact match with a model, the code corresponding to the model for which the Hamming distance is minimum is assigned to the pixel. This coding process produces a 4 bit/pixel image where edges are partially regularized and restored.

After such a morphological coding the frame is ready for the subsequent matching step.

2.2 Matching

In the matching step, a comparison between *New-image* and *Old-image* is made, to determine if and how much the edges extracted have been displaced.

To define the matching problem operatively we can consider, for example, an easy case where, in two consecutive images, all objects are simply translated with respect to the observer. In this case, each point $P(x, y)$ belonging to *Old-image* corresponds to the point $P(x + \alpha, y + \beta)$ belonging to *New-image*, where (α, β) are the x and y components of the translation on the image plane; the matching problem consists in the detection of the two components (α, β).

In the real case the problem is more complicated because:

(1) Generally the vectors of components (α, β) are distributed in several directions and do not have a predictable magnitude.

(2) Occlusions (occurring when an object hides another object) and image clipping (occurring when objects disappear, partially or totally, in the new frame) prevent some points from being matched.

(3) In many cases objects present regularities that make the association process ambiguous. In such cases, for each point belonging to *Old-image* there may be two or more points belonging to *New-image* which match.

(4) Noise, as well as optical and perspective distortions, may alter object appearance over time.

Points 1 and 3 can be tackled successfully by the proposed local algorithm. Also, the effects of noise and of some small perspective distortions can be limited by the morphological coding operation, while point 2 requires a global analysis of the scene, which cannot be made using a local approach.

The algorithm analyzes the global motion in eight phases, one for each direction (North, North-East, East, etc.). In other words, eight variants of one unidirectional matching detector are used. The basic version requires a

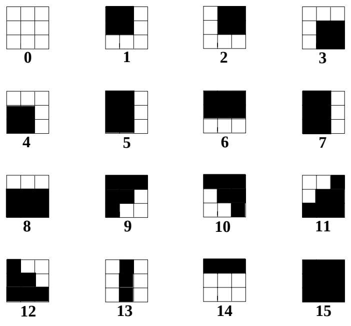

Figure 2. The 16 models used for the morphological coding of edges.

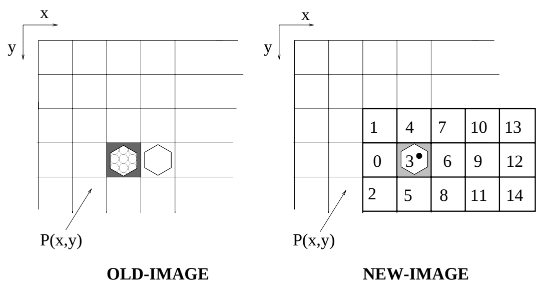

Figure 3. Example of the grid used to detect translations to the right, and the corresponding cell indices.

comparison between one point belonging to *Old-image* and the fifteen points that are closest to the corresponding pixel of *New-image* in the region of the grid that extends in a prefixed direction. For example, in the detector used to reveal translations to the right, pixel are compared as showed in Figure 3. If we consider the generic pixel $P(x, y)$ belonging to *Old-image*, such a pixel is compared with the following pixels belonging to *New-image* (refer to Figure 3): $P(x, y), P(x, y-1), P(x, y+1), P(x+1, y), P(x+1, y-1), P(x+1, y+1), \ldots, P(x+n, y), P(x+n, y-1), P(x+n, y+1)$.

The following outcomes may occur:

(a) There is only one pixel in the grid that matches.

(b) There are more than one pixel in the grid that match.

(c) No pixel in the grid matches.

The process actually detects the closest match (which, in a sequential implementation, would mean that the process is stopped as soon as a candidate is found), so the matching algorithm treats case (a) and case (b) in the same way. Case (c) assumes by convention that the displacement is zero. At the end of the scan the position with respect to the origin of the closest candidate (the dotted cell in the grid of Figure 3) corresponds to the displacement to be detected. This solution assumes implicitly that the correct detection is the one that corresponds to the smallest displacement, as the hypothesis is made that object velocity is low with respect to the rate of the matching analysis.

Eight versions of this matching detector (applied along different directions) produce eight directional fields which, if vectorially added, permit the computation of the global field.

3 A case study

The reference architecture used in our experiments is the CAM-8 [10], a special architecture for the implementation of uniform cellular automata. The CAM-8 is an indefinitely-scalable multi-processor architecture aimed at the fine-grained modeling of spatially extended systems. Physically, the CAM-8 is a three-dimensional mesh of modules operating in lockstep on pipelined data.

In brief, the CAM-8 maps a n-dimensional space ($n >= 1$) onto a grid of evenly-spaced cells. Each cell holds a 16-bit word which represents the cell statuses, so 2^{16} different cell status are possible. The evolution of the system is computed at discrete time steps: at each time step the state of each cell is univocally determined by the cell status in the previous step. The status word may be composed by collecting bits from the cell itself and from any other cell in the space. The present cell status is used to address a look-up table (LUT) of size 2^{16}. The content of the corresponding LUT location will determine the future cell status.

One processing step consists of a serial scanning of the whole set of cells according to the uniform CA paradigm where, in each step, the same operation is performed in the whole cell space. The time required for a single interaction Cell - LUT - Cell in which all cells in the grid change status is the basic time unit to be considered for time complexity analysis. To underline this property it is named *scan* or *step*.

The CAM-8 model used in this work includes eight base modules, each of which is a complete system (memory space, LUT, etc.), which can work in parallel (each module works on a sub-section of the same universe) or independently (each module computes a different operation in its own space).

The first operation that is performed is the acquisition of *New-image* through a direct link between a digital B/W CCD camera and the cell memory space. A bi-dimensional memory space is defined and the whole image is transferred from the camera to the CAM-8 memory in a single “scan”.

The phase of image pre-processing requires one scan and one LUT. The edge extractor which implements Robert’s operator requires two scans and two LUTs, while morphological edge coding requires only one scan and one LUT.

The matching step is more complicated than the previous one. The eight directional detectors are implemented one for each of the eight modules of the CAM-8 and are operated in parallel. A single LUT is used to implement a step counter (*Step Index*, a 4-bit word) and a cell register (*Cell Index*, a 4-bit word). The counter is initialized at 14 (the number of pixels in the region in which matching takes place) and goes down to 0 with a rate of one for each scan.

During a single scan a given cell of *Old-image* and the cell of *New-image*, whose position is encoded in the *Step Index*, are compared; if they match, the *Cell Index* is assigned the value of the *Step Index* (as shown in Figure 4. Doing so, when the *Step Index* is zero only the last match (for each cell of *Old-image*) is recorded, that is the nearest to the grid origin. This process requires 15 scans and only one LUT. The result of the matching is stored in the cells using the *Cell Index*.

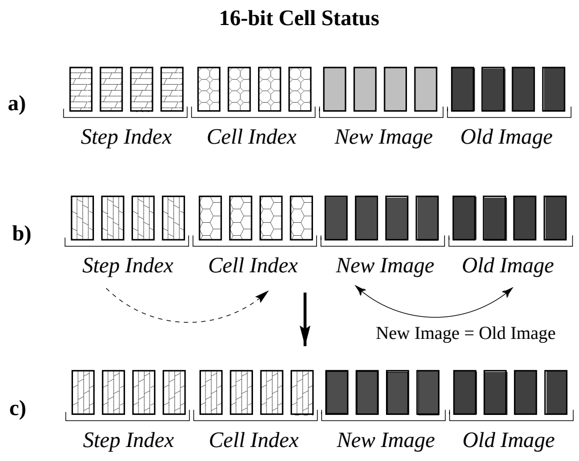

Figure 4. The cell status in a step of the pixel matching cycle: a) the description of the cell status; b) the cell status when the pixel of the new image matches with the old pixel c) the cell status after the assignment of Step Index to Cell Index.

The next step performs the conversion from the grid index to the two components of the corresponding vectors. In this step it is possible to perform a number of operations to enhance algorithm robustness; for example, it is possible to remove all vectors whose module is below a given threshold, or select a specific set of vectors, or rescale them within a selected range. To perform the conversion, one step and one LUT are needed. This is the last step of the optical flow detection, and so it also includes the *New-image* to *Old-image* copy operation.

Finally the optical flow results are processed by the host for visualization or further processing.

4 Performance evaluation

The preliminary tests were aimed at verifying system performance in a well-known condition, to make a subsequent evaluation of its robustness possible. In these experiments an image sequence was used in which the objects have specified geometric characteristics with uniform surfaces and are observed in conditions of constant brightness.

Several experiments on laboratory image sequences were performed. Such artificial sequences represent a few seconds of motion simulation. As an example, we consider the oblique translation of a little boat from left to right (the first and the last frames of the sequence are showed in Figure 5).

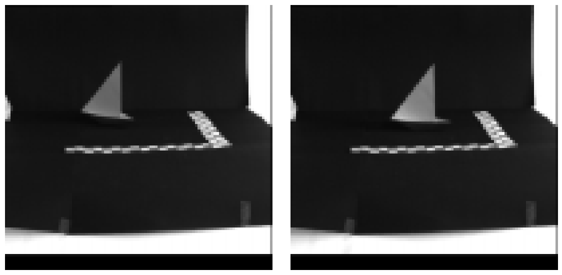

Figure 5. First and last frames of a sequence in an indoor and controlled environment.

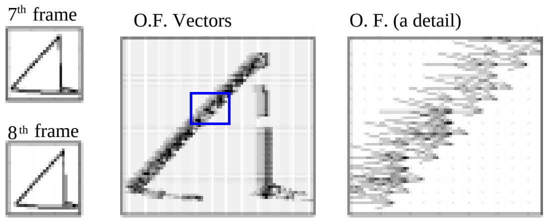

Figure 6. Optical-flow trace computed between the 7^{th} and 8^{th} frames of the sequence of Figure 5 together with the output of the Robert's filter applied to the 7^{th} and 8^{th} frames, respectively.

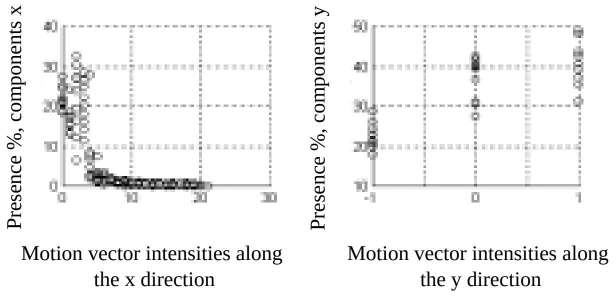

Figure 7. Optical-flow component population for the sequence of Figure 5.

Figure 6 shows the optical flow trace computed between two adjacent frames (the 7^{th} and 8^{th}), i.e., the instantaneous reconstruction of the movement. Figure 7 reports the plot of the population of optical flow components over the whole sequence. Such a plot can be used to assess the system stability over time as regards motion detection, since, in this example, the motion components are constant.

After the first results obtained on limited and controlled test beds, the method was applied to uncontrolled and noisy environments.

As an example, we consider the sequence in which a car moves from right to left about 10m away from the videocamera. The environment comprises both uniform regions and other irregularly-shaped areas. The sequence includes 16 frames, of which the first and last are showed in Figure 8.

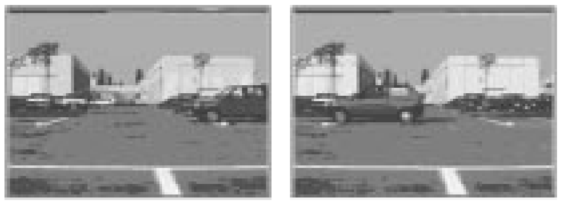

Figure 8. The first and last frames of an outdoor sequence.

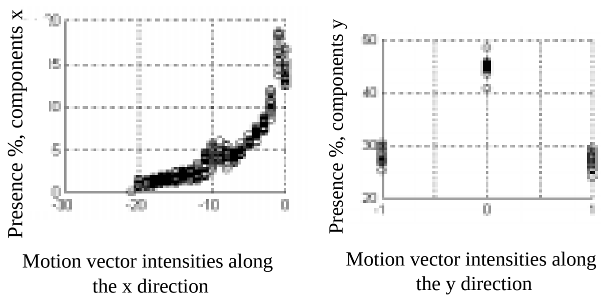

Figure 9. Optical-flow component population for the sequence of Figure 8.

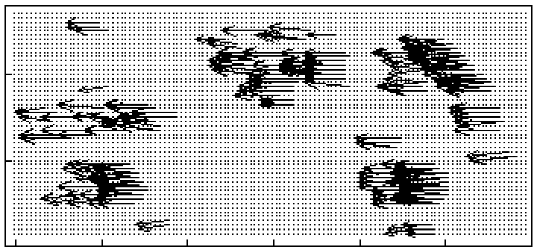

Figure 10. Optical-flow trace computed between the 14^{th} and 15^{th} frames of the sequence of Figure 8.

The plot of the optical-flow components is showed in Figure 9. In the two adjacent images the car displacement corresponds to a left shift of about 9-10 pixels. However, the diagram shows a dominance of vectors of smaller extension: this is due to the strong presence of noise in the environment, which causes a high number of false movements to be detected. To reduce this noise we removed those vectors that were smaller than an empirically-set threshold; doing so, the actual car movement could be reconstructed as showed in the detail of the optical flow image computed between the 14^{th} and 15^{th} frame of the sequence (Figure 10).

The results achieved show that the optical-flow computation algorithm, when implemented on a dedicated cellular-automata architecture such as the CAM-8, is very efficient, because all operations can be executed without any information exchange with the host: that is, the algorithm proposed is fully cellular. Moreover, it requires very limited memory resources (only 16 bits per pixel). Its temporal complexity can be deduced, using the CAM-8 as reference, by adding up the time slices (*steps*) required to perform the whole sequence of operations. In Table 1, we report the execution times for a single optical flow detection for different image sizes and number of CAM-8 modules, which can update the 16-bit cells status at a rate of 25 Megacells/s per module. It is worth noting that the execution time of the algorithm is almost independent of image size if the number of CAM-8 modules is sufficient to contain the whole image. One should also remember that the CAM-8 relies on late-eighties/early-nineties technology. Therefore, on architectures based on more recent technology, performances could be much better.

image resolution (pixel)	*time for 1 module (s)*	*time for 8 modules (s)*
256 × 256	0.0960	0.0708
512 × 512	0.2544	0.0744

Table 1. Algorithm execution time.

5 Concluding remarks

This paper presented an algorithm for feature-based optical flow detection. The proposed algorithm satisfies the requirements of many practical robot applications. For real-world robotic applications, an optical flow computation algorithm must find a good trade-off between temporal constraints and a satisfactory degree of reliability. The execution time of the algorithm described in this paper makes it possible for a robot to produce “just in time” responses to changes in the environment. Moreover, the experimental tests gave good performances in terms of motion detection both in indoor controlled environments and in outdoor noisy environments.

In this method only two images are analyzed at the same time; the use of previous results is not required when considering a sequence of frames. We are investigating the possibility of correlating the optical flow detections in subsequent frames, which may be useful when solving some ambiguous situations and enhance algorithm robustness.

Acknowledgments

This work is partially supported by the “Ministero dell’Università e della Ricerca Scientifica e Tecnologica” under grant Murst 40% “CERTAMEN”.

References

[1] G. Adorni, M. Gori, and M. Mordonini. Just-in-time sign recognition in image sequences. *Journal of Real-Time Imaging*, 5(2):95–107, 1999.

[2] G. Adorni and A. Poggi. Route guidance as a just-in-time multiagent task. *International Journal of Applied Artificial Intelligence*, 10(2):95–120, 1996.

[3] S. Barnard and W. Thomson. Analysis of images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2(4):333–340, November 1980.

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

[5] M. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[6] A. Del Bimbo and P. Nesi. Optical flow estimation on the connection machine 2. In *Workshop on Computer Architecture for Machine Perception*, pages 267–274, New Orleans, LA, 1993.

[7] R. Dutta and C. Weens. Parallel dense depth from motion on the image understanding architecture. In *Proceedings of CVPR*, pages 154–159, 1993.

[8] L. Hongche, H. Tsai-Hong, M. Herman, T. Camus, and R. Chellappa. Accuracy vs. efficiency trade-offs in optical flow algorithms. *Computer Vision and Image Understanding*, 72(3):271–86, 1998.

[9] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, 1981.

[10] N. Margolus. CAM-8: a computer architecture based on cellular automata. In A. Lawniczak and R. Kapral, editors, *Pattern Formation and Lattice-Gas Automata*. American Mathematical Society, 1994.

[11] P. Nesi, A. Del Bimbo, and D. Ben-Tzvi. A robust algorithm for optical flow estimation. *Computer Vision and Image Understanding*, 62(1):59–68, 1995.

[12] S. Smith. Asset-2: Real-time motion segmentation and object tracking. *Journal of Real Time Imaging*, 4(1):21–40, 1997. Special issue on real-time motion analysis.

[13] T. Toffoli and N. Margolus. *Cellular Automata Machines-A New Environment for Modeling*. MIT Press, Cambridge, MA, 1986.

[14] F. Wahl. *Digital Image Signal Processing*. Artech House, Boston, MA, 1987.

[15] Q. Wu. A correlation-relaxation-labeling framework for computing optical flow-template matching from a new perspective. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(9):843–853, 1995.