Proc. of SPIE'92. Applications of Artificial Intelligence Machine Vision and Robotics. 22-24/04/92, Orlando.
92-C

Parallel VLSI-based Architecture for Multi-motion Estimation

J.D. Legat, J.P. Cornil, D. Macq, M. Verleysen

Microelectronics Laboratory
Electrical Engineering Department
Université Catholique de Louvain
B-1348 Louvain-la-Neuve
Belgium

ABSTRACT

This paper describes a new parallel architecture dedicated to multi-motion estimation. The input image is scanned by a standard video camera with 256 grey levels. Motion computing is based on the optical flow determination. Some constraints are proposed to allow multi-motion evaluation. The algorithm will be presented and the main features of a 1-D systolic architecture which is based on a custom VLSI chip will be given. This architecture allows a real-time implementation of the multi-motion estimation algorithm.

1. INTRODUCTION

Motion estimation of objects or people from image sequences plays a key role in early vision processing [1]. It is used in dynamic image segmentation and recognition. Relative motion allows mobile robot to navigate quickly and efficiently through the environment [2].

There exist two basic algorithms for determining motion [3]. The first is based on the estimation of special features in the image which are then matched from image to image. This method supposes that the image is first analyzed carefully before computing motion. Psychophysical tests suggest that it is not the case in the human system. The second algorithm uses local gradients of the image brightness to compute the optical flow. In general, the optical flow and the true 2-D velocity field differ. Nevertheless, if strong enough gradients exist, the optical flow will be a good approximation of the velocity field.

Through many types of optical flow estimation techniques have been developed [3,4], most of them are restricted to global motion estimation [5]. However, most of the real-life applications require multi-motion evaluation.

This paper describes the architecture and the implementation of a parallel processor dedicated to motion estimation which is based on a custom VLSI chip. This architecture allows a real-time implementation of a multi-motion algorithm based on the optical flow estimation. The general configuration is represented at figure 1. The input image is scanned by a standard video camera with 256 grey levels and the motion evaluation system is directly connected to the camera output.

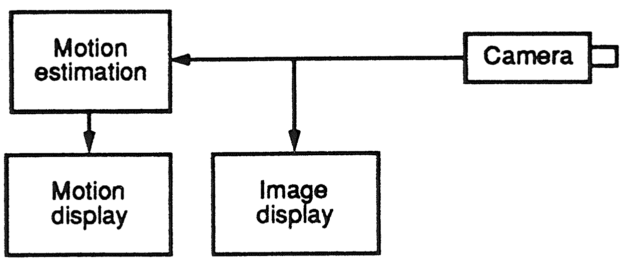

Figure 1 : General configuration of multi-motion system

2. MULTI-MOTION EVALUATION

Following Horn and Schunk [6], we denote the image brightness at the point (x,y) in the image plane at time t by I(x,y,t). When a pattern moves, the brightness of a particular point in the pattern is constant, so that :

$$\frac{dI}{dt} = 0$$

Using the chain rule of differentiation and defining the velocity V as $(v_x, v_y) = (dx/dt, dy/dt)$, we obtain a single linear equation in two unknowns, v_x and v_y :

$$\frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y + \frac{\partial I}{\partial t} = 0$$

This equation by itself is not sufficient to determine the velocity flow. It only defines a constraint line in velocity space that has the same orientation as does the edge of the moving pattern in physical space. This ambiguity is known as the "aperture problem". If we suppose that there is only one moving pattern, each edge of the moving object generates a constraint line. The intersection of these lines determines the actual velocity .

If many objects can move independently, it will not be possible to determine the velocity without introducing additional constraints. We can suppose that neighboring points of a moving object have similar velocities and the velocity field in the image varies smoothly. Based on this smoothness assumption, we introduce the three following constraints :

1. If a moving object at time t has the velocity v(t), it will have at time t+1 a velocity v(t+1) so that :

$$v(t) - \Delta v < v(t+1) < v(t) + \Delta v$$

In the velocity space as illustrated in figure 2, if we have a velocity v(t) defined by the intersection of 2 constraint lines, at time t+1, the velocity v(t+1) has to be included inside a circle having as center v(t) and as radius Δv .

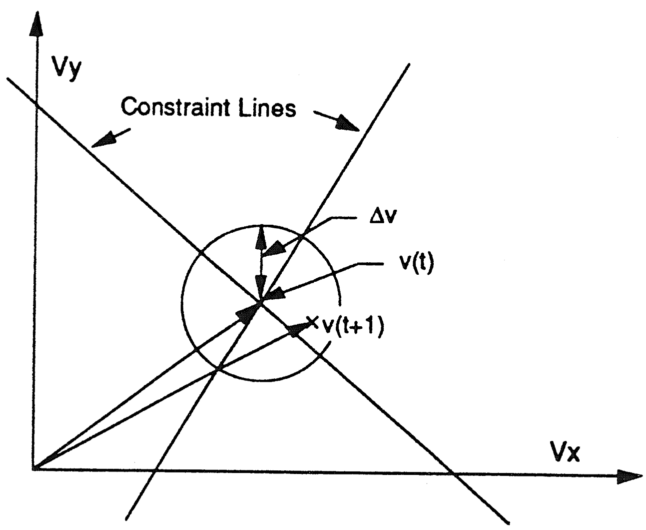

Figure 2 : Constraint 1 for multi-motion computing.

2. If this moving object at time t is included in a rectangular boundary having the coordinates ((x0(t), y0(t)), (x1(t),y1(t))), it will be included at time t+1 into a rectangle having the coordinates ((x0(t+1), y0(t+1)), (x1(t+1),y1(t+1))) so that (figure 3) :

$$x0(t+1) = x0(t) + ((v_x(t) - \Delta v) \,.\, \Delta t) \quad \text{and} \quad y0(t+1) = y0(t) + ((v_y(t) - \Delta v) \,.\, \Delta t)$$
$$x1(t+1) = x1(t) + ((v_x(t) + \Delta v) \,.\, \Delta t) \quad \text{and} \quad y1(t+1) = y1(t) + ((v_y(t) + \Delta v) \,.\, \Delta t)$$

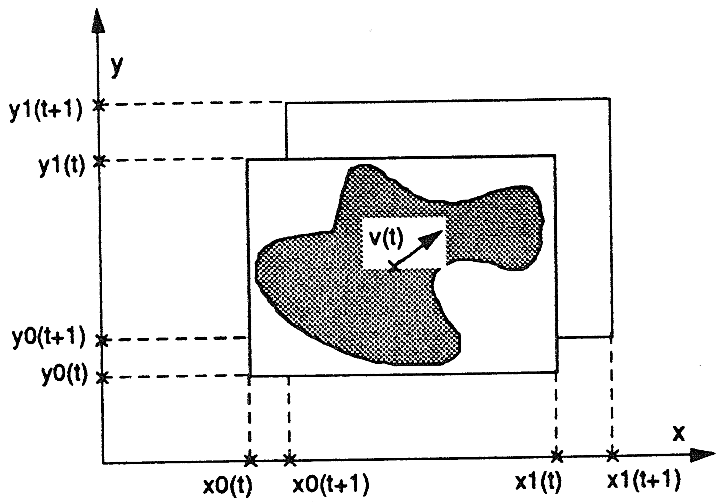

Figure 3 : Constraint 2 for multi-motion computing.

3. Only one new motion can appear at the same time. Indeed, if 2 objects can set themselves in motion at the same time, the system has no way to separate them because their initial velocity is null and their initial boundary rectangle is all the scene. In practice, this constraint is not very restricting due to the fact that the image frequency is high and thus the probability to have 2 new motions exactly at the same time is quite low.

By using these hypotheses, it is now possible to associate each constraint line with a particular motion and to efficiently initiate new motions. This method is illustrated by 2 simple examples (figure 4 and figure 5) where two objects, a rectangle and a triangle, are moving. In the first, the objects are moving in different directions : the rectangle has 2 constraint lines and a velocity V1 at the intersection of the 2 lines, the triangle has 3 lines and the velocity V2. The constraint 1 which allows V1 and V2 to fluctuate inside their respective circle, is sufficient to detect the 2 motions.

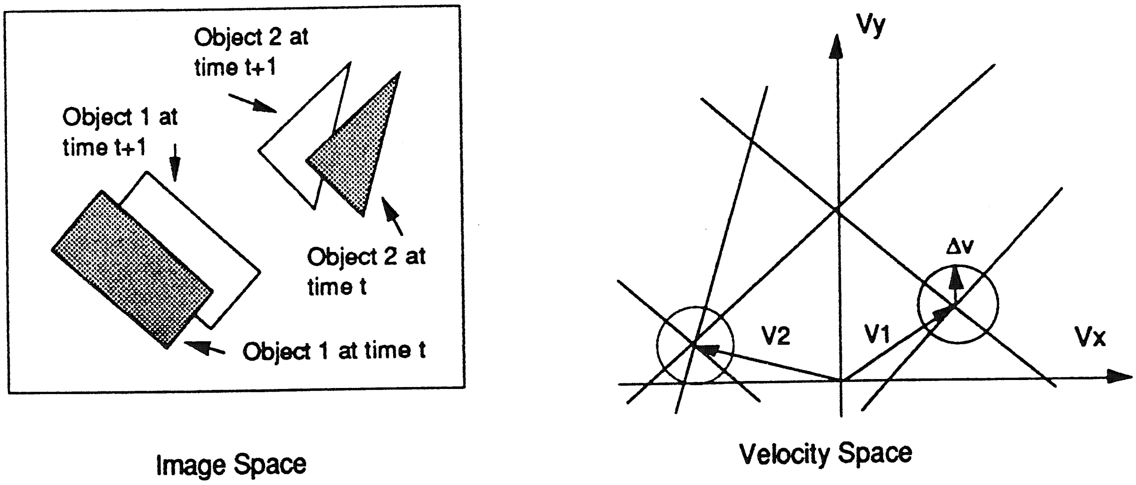

Figure 4 : Example of multi-motion computing

In the second example (figure 5), the 2 objects have a similar speed. In the velocity space, the 2 circles are overlapping and the constraint 2, which defines a boundary rectangle for each moving object, is necessary to be able to separate the 2 motions.

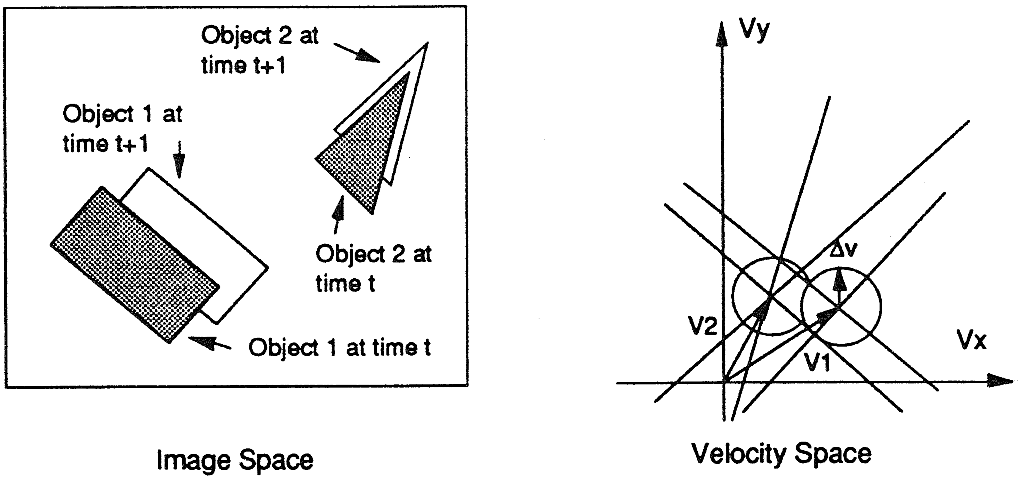

Figure 5 : Example of multi-motion computing

The algorithm has been successfully simulated on synthetic and some natural time-varying images. There are of course some limitations. The main problems occur when there are too many noisy gradients or when the moving objects are crossing or are overlapping. In this case, the boundary rectangle is not useful and during the crossing time the result represents more or less the average of the 2 motions. If an object has different movements, the system will fail or will only detect the main motion.

3. PARALLEL ARCHITECTURE

To obtain a real-time implementation of the multi-motion algorithm, a parallel architecture based on an 1-D systolic array has been developed (figure 6). Each node of the systolic array handles a specific motion and the array has been designed in such a way that it filters the moving parts of the image. The first node of the array receives the input image from the camera, computes the motion 1, generates an output image without the moving object 1 and transfers this image to the node 2. The second node computes the motion 2 and so on. At the end of the array, we have an image without moving object if the number of nodes is equal or superior to the number of movements to detect.

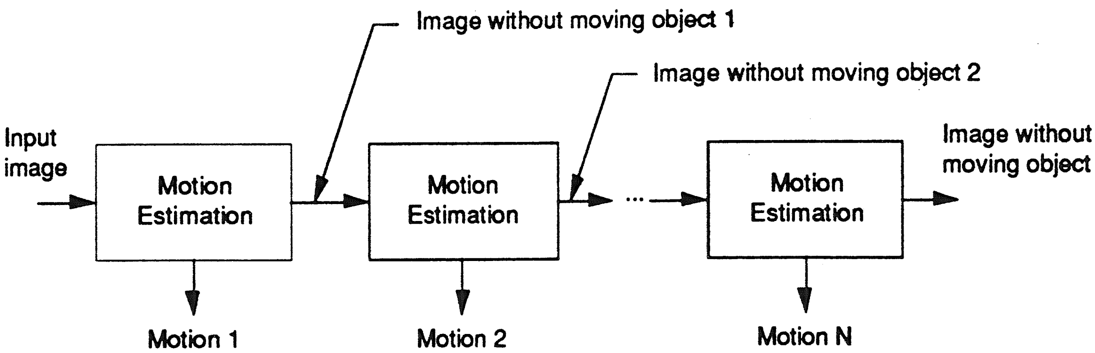

Figure 6 : Systolic array architecture.

A global view of all the system is shown on figure 7. Each node of the systolic array is a full processor. The camera having a serial output, the processing is done pixel by pixel. The pixel is transfered to the first processor plus some other valuers generated by a delay logic. The pixel is processed by the first processor and then transmitted to the second one. A disable signal is used to filter the pixels. This signal is always true at the input of the processor and becomes active if the pixel has been associated with the current moving object. At the last node, the disable signal acts as an overflow signal indicating if the number of motions to detect is superior or not to the number of nodes. Each processor works synchronously and has the same architecture.

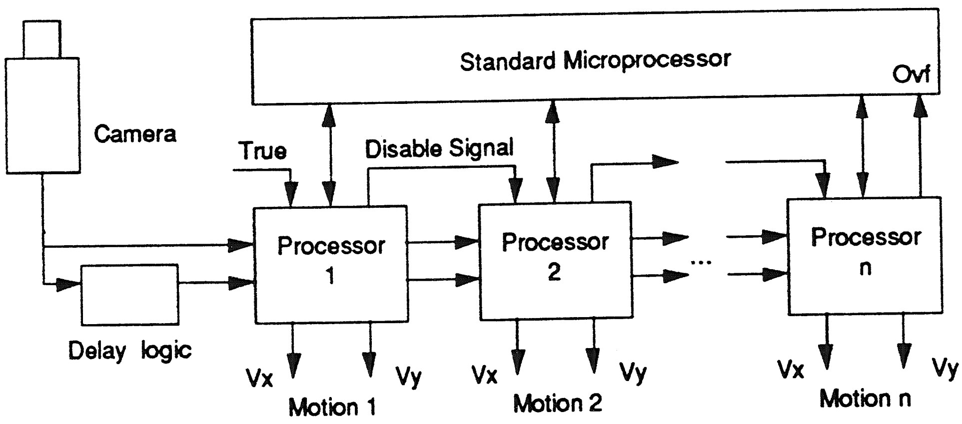

Figure 7 : systolic array architecture

The delay logic is a simple logic based on FIFO memory which is used to simultaneously generate the different values of the intensity needed by the multi-motion estimation algorithm (figure 8).

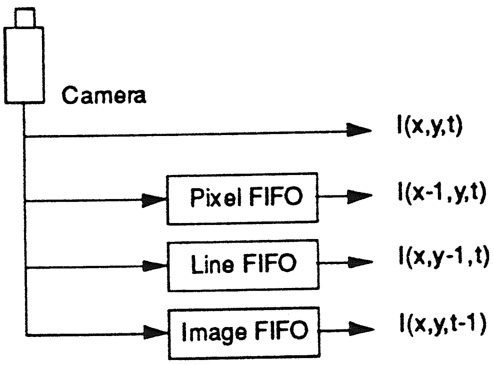

Figure 8 : delay logic

A standard microprocessor controls at a high level all the processing. Its main functions are the following :

- Receive from each processor the Vx and Vy components of the velocity, the coordinates of the boundary zone and some other parameters such as the number of pixels which have been associated to this motion.
- Eliminate the parasitic motions.
- Sort the motions. Due to the filtering process of the systolic array, the order of motion processing is not without importance. A good choice may solve problems when two moving objects are crossing.
- Calculate the zone where the moving object has to be at time t+1.
- Re-program each processor by sending the new coordinates (x0(t+1), ...), the velocity components Vx and Vy and some parameters (Δv, ...).

4. VLSI IMPLEMENTATION

The elementary processor of the systolic array is implemented in a custom integrated circuit. The main tasks of the VLSI circuit are represented on the flow-chart of figure 9. The first step is to receive the pixel from the previous processor. The circuit then tests the constraints 1 and 2. If they are not satisfied, the pixel is directly transmitted to the next processor. Otherwise, the velocity and the boundary zone are updated.

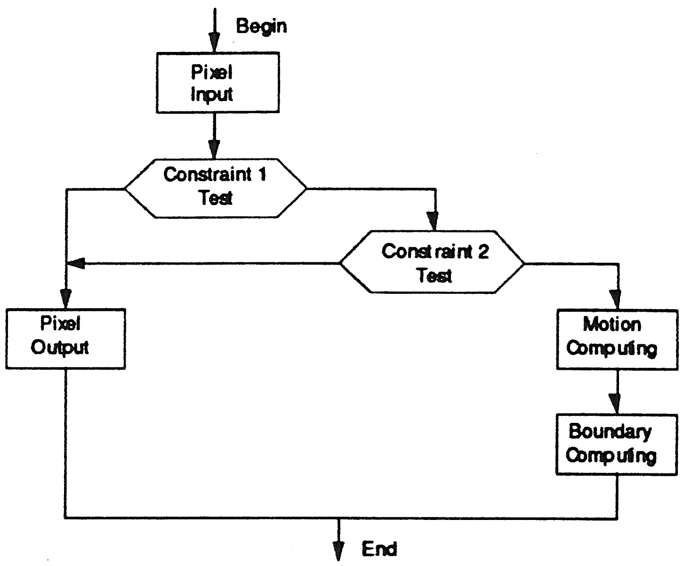

Figure 9 : circuit functions

As illustrated in figure 10, the chip consists of 9 processing units : the input pixel unit (IPU) which receives the pixel brightness and some other values from the previous processor (or from the camera for the first processor), the parameter computing unit (PCU) which determines common parameters used by the other blocks, VXU and VYU which compute the velocity in the X and Y directions, BXU and BYU which estimate the boundary of the moving object in the X and Y directions, the output pixel unit (OPU) which transfers the pixel to the next processor of the array, the microcoded control unit (MCU) and the microprocessor interface unit (MIU) which controls all interactions between the processor and the standard microprocessor.

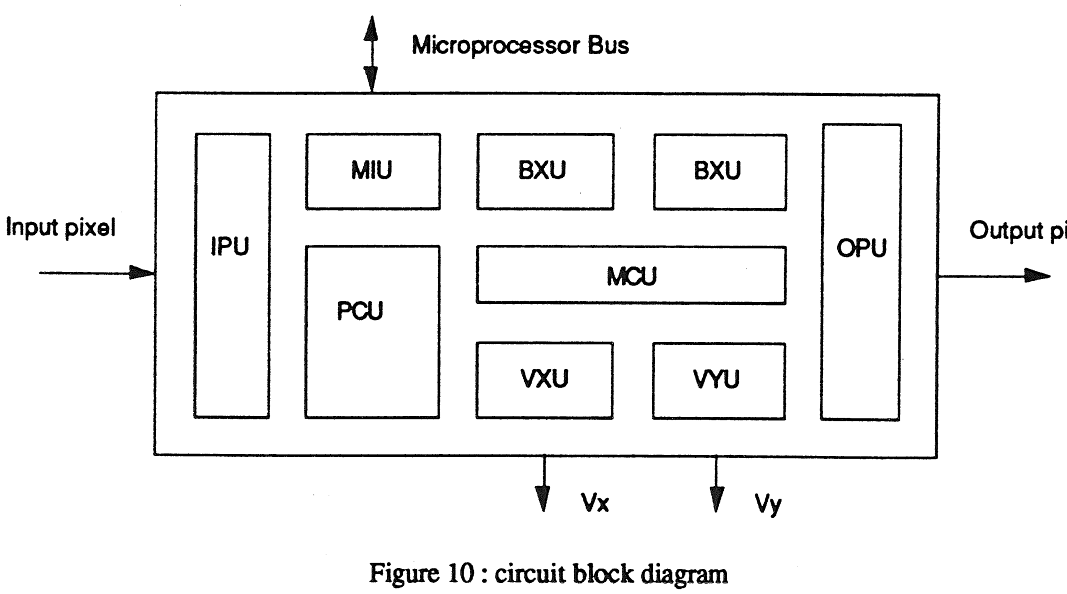

Figure 10 : circuit block diagram

Due to the real-time requirement, a very high parallelism and pipe-line has been achieved. Only four clock cycles are needed by the processor to perform all the operations associated with one pixel (parameters computing, velocity updating, boundary checking, ...).

The custom integrated circuit is designed in a standard CMOS technology. The die size is approximately 60 mm^2. The chip has been designed to have a clock frequency of 40 MHz which will allow to have a pixel frequency of up to 10 MHz.

5. CONCLUSION

An algorithm using the optical flow estimation and the smoothness constraint for multi-motion computing has been described. A parallel architecture allowing a real-time implementation in a standard video camera environment has been presented. This architecture is based on a custom integrated circuit which integrates the full specifications of the elementary array processor.

6. REFERENCES

1. D.G. Stavenga and R.D. Hardie, "Facets of Vision," Springer Berlin Heidelberg 1989.
2. J.M. Pichon, C. Blanes and N. Franceschini, "Visual guidance of a mobile robot equipped with a network of self-motion sensors," *Mobile Robots IV* , SPIE vol. 1195, pp 44-53, 1989.
3. J. Hutchinson, C. Koch, J. Luo and C. Mead, "Computing Motion Using Analog and Binary Resistive Networks," *Computer*,, pp 52-63, March 1988.
4. J. Wu, R. Brockett and K. Wohn, "A Contour-based Recovery of Image Flow : Iterative Method," *CVPR'89 - IEEE Conference on Computer Vision and Pattern Recognition*, San Diego, pp 124-129, 1989.
5. J. Tanner and C. Mead, "Optical Motion Sensor", Analog VLSI and Neural Systems, Addison-Wesley, 1989.
6. B. Horn and B. Schunck, "Determining Optical Flow," *Artificial Intelligence*, vol. 17, pp 44-53, 1981.