Parallel issues of an optical flow estimator based on a quasi-interpolant operator

E. FRANCOMANO*^, C. LODATO^, S. LOPES^, A. TORTORICI*^  
* Dipartimento di Ingegneria Automatica ed Informatica  
Università di Palermo  
Viale delle Scienze, 90128 Palermo  
ITALY  
^CEntro studi sulle Reti di Elaboratori (CERE) - CNR  
Viale delle Scienze - 90128 Palermo  
ITALY

Abstract: - This paper addresses the parallel issues of a novel method entirely developed by the authors devoted to estimate the velocity vector field of the apparent motion of brightness patterns usually referred as optical flow. Optical flow plays a fundamental role in many real-time computer vision applications. In this paper the flow field has been performed by adopting a bivariate quasi-interpolant operator based on centered cardinal B-spline functions onto a non linear minimizing algorithm. The solving model involves massive computational amount so that high-performance system need to rapidly proceed in the computations. Therefore, in the following a parallel analysis of the computational scheme, oriented to distributed multiprocessors platforms, has been carried out. The process has shown to be synchronous, with good tasks balancing and requiring few amount of data transfer.

Key-Words: - Computer vision, Optical flow, B-spline functions, Quasi-interpolant operator, Levenberg-Marquardt algorithm, Distributed multiprocessors.

1 Introduction

The most common approach for the analysis of visual motion is based on the computation of an optical flow field, i.e. a velocity field composed of vectors describing the instantaneous velocity of image elements. Optical flow computation is currently motivated by many applications in computer vision and it is an important prerequisite when visual tasks, such those involving a robot autonomous locomotion and its physical interaction with the environment, are required.

Optical flow is successfully used to perform object segmentation, to compute the focus of expansion and time-to-collision, to compute stereo disparity, to measure blood flow and heart-wall motion [1,2,3,4,5].

In this paper a non linear minimizing technique [6] involving a bivariate quasi-interpolant operator has been used to perform optical flow estimates. The quasi-interpolant adopted composed with a timely scaling process gives rise to a multiresolution coarse-to-fine process [7,8].

As in theory verified [7], improvements in the solution can be performed by increasing the granularity. Furthermore, at each level of the multigrid process the non linear minimizing technique involves iterations to better generate velocity values at the current grid and more computations than those implicated at the previous level are required. Moreover, the mainly processing tasks involved in the studied model require local operators which involve regular and repetitive estimations computationally intensive. As a consequence it is advisable to proceed in the computation by employing multiprocessor platforms to rapidly obtain sufficiently accurate results.

In this paper the parallel features of the solving model have been investigated. Studies on data distribution, computational demanding and communications requirement have been carried out. The remainder of the paper is organized as follows. Section 2 briefly reviews selected preliminaries on a quasi-interpolant operator generated by using cardinal centered B-spline functions and the bivariate formulation used has been reported. Section 3 deals with the computational algorithm developed to estimate the velocity vector field. Section 4 reports studies on the parallel features of the solving scheme oriented to distributed multiprocessors platforms.

2 Quasi-interpolant Operator

In this paper a linear spline operator that maps \( C_P(\mathbb{R}) = \{ f \in C(\mathbb{R}) : \sup|f(x)| < \infty \} \) to the centered cardinal B-spline functions space \( V_0^n \) :
\[
(Q_f)(x) = \sum_{n=2}^{\infty} (E_n f)(l) N^m\left(x + \frac{m}{2} - l\right)
\]  \hspace{1cm} (1)

has been taken into account where \(N^m(x)\) is the \(m\)th order cardinal B-spline functions with \(\text{supp} \ N^m = (0,m)\) and:

\[
(\hat{E}_n f)(l) = \sum_{j \in Z} \delta_{(\cdot - j)} f_j \quad l \in Z
\]  \hspace{1cm} (2)

is a convolution operator [7,8].

The (1) is a bounded linear and local operator, in the sense that each value \((\hat{E}_n f)(l)\) depends only on the \(f_j\) value in a neighborhood of \(j = l, \forall l\).

The quasi-interpolant \(Q\) operator supplies simple and computationally efficient schemes for constructing cardinal spline approximants providing the highest order of approximation [7,8]. Moreover, a multiresolution process can be carried out: by considering the data sequence in the spline space \(V^m\), a data sampling in the spline space \(V^m\) can be derived by fixing the sampling period be as small as \(h = 2^{-r}\).

Fig.1 Three levels of the coarse-to-fine process

By composing the scaling process \((s_h f)(x) = f\left(\frac{x}{h}\right)\) and the quasi-interpolant operator \(Q\), namely:

\[
Q^h = s_h \circ Q \circ s_h^{-1}
\]  \hspace{1cm} (3)

an approximation formula can be generated. By considering \(m=4\) and \(s=1\) the bivariate formula can be written as:

\[
(Qf)(x, y) = \sum_{l \in Z, k \in Z} \sum_{j \in Z} \delta_{(\cdot - j)} f_{j} N^4(x-2l+2) \quad N^4(y-2k+2)
\]  \hspace{1cm} (4)

By fixing \(x = q\) and \(y = r\), \(q, r \in Z\), the only terms in the summations correspond to the indexes \(l\) and \(k\) verifying the following conditions: \(q-1 \leq l \leq q+1\) and \(r-1 \leq k \leq r+1\).

The formula (4) can be rewritten requiring only data information in even points giving the evaluation in odd points and implicitly performing one resolution level that is:

\[
(Qf)(x, y) = \sum_{l \in Z, k \in Z} \sum_{j \in Z} \delta_{(\cdot - j)} f_{j} N^4(x-2l+2) \quad N^4(y-2k+2)
\]  \hspace{1cm} (5)

where:

\[
c_{-11} = c_{11} = c_{-1-1} = c_{1-1} = \frac{1}{576}
\]

\[
c_{01} = c_{10} = c_{-10} = c_{0-1} = -\frac{13}{288}
\]

\[
c_{00} = \frac{169}{144}
\]

\[
c_{-j} = 0 \quad \text{otherwise}
\]

\[
d_{-21} = d_{11} = d_{-2-2} = d_{1-2} = \frac{1}{144}
\]

\[
d_{-11} = d_{01} = d_{-1-2} = d_{0-2} = d_{-j-2} = d_{j-2} = \frac{7}{144}
\]

\[
d_{00} = d_{0-1} = d_{-11} = d_{-10} = \frac{49}{144}
\]

\[
d_{ij} = 0 \quad \text{otherwise}
\]  \hspace{1cm} (6)

\[
g_{-11} = g_{11} = g_{-1-2} = g_{1-2} = \frac{1}{288}
\]

\[
g_{01} = g_{0-2} = -\frac{13}{144}
\]

\[
g_{10} = g_{1-1} = g_{-11} = g_{-10} = -\frac{7}{288}
\]

\[
g_{00} = g_{0-1} = \frac{91}{144}
\]  \hspace{1cm} (7)
3 Optical Flow Estimator

Given a sequence of images, the function $I(x,y)$ is the image brightness over time computed at pixel $(x,y)$. Optical flow is defined as the displacement vector $\hat{v}(x,y) = (u(x,y),v(x,y))$ resulting from the apparent motion of brightness patterns in the images. The aim is to recover, at each pixel, the components $u$ and $v$. A solution to this problem consists in minimizing a Sum of Squared Differences (SSD) formula involving $I(x,y)$. By taking into account a sequence composed by only two frames the SSD relation can be expressed as:

$$E = \sum_x \sum_y |I(x_i + u_i, y_j + v_j) - I_0(x_i, y_j)|^2$$  \hspace{1cm} (10)

where $u_i = u(x_i, y_i)$ and $v_j = v(x_j, y_j)$ are computed in all image pixels. The idea is to represent the displacement field by using a local approximation model such as a quasi-interpolant based on the operator $Qf$ previously described.

Let $\hat{u}_{mn}$ and $\hat{v}_{mn}$ be initial values for the displacement field. A first approximation of the displacement field can be expressed as:

$$u(x,y) = (Qu)(x,y)$$  \hspace{1cm} (11)

$$v(x,y) = (Qv)(x,y)$$  \hspace{1cm} (12)

The cost function (10) must be minimized with respect to $\hat{u}_{mn}$ and $\hat{v}_{mn}$ and the modified Levenberg-Marquardt iterative non linear minimization technique can be adopted [6,10] in order to obtain an increment of $\hat{\nabla}$ the initial values $\hat{u}_{mn}$ and $\hat{v}_{mn}$. By indicating with:

$$e_{ij} = I_1(x_i + u_i, y_j + v_j) - I_0(x_i, y_j)$$  \hspace{1cm} (13)

where $u_i$ and $v_j$ are the values computed with (11) and (12), and with:

$$g_{ij} = 0 \quad \text{otherwise}$$  \hspace{1cm} (8)

$$h_{-2l} = h_{l+1} = h_{-l-1} = h_{l-1} = \frac{l}{288}$$

$$h_{-20} = h_{10} = \frac{-144}{144}$$

$$h_{-11} = h_{10} = h_{-l-1} = \frac{7}{288}$$

$$h_{10} = h_{-10} = \frac{91}{144}$$

$$h_{ij} = 0 \quad \text{otherwise}$$  \hspace{1cm} (9)

$$G_{ij} = \nabla I_1(x_i + u_i, y_j + v_j)$$  \hspace{1cm} (14)

the intensity gradient at the displaced position for the pixel $(x_i, y_j)$, the gradient of the cost function can be computed as:

$$\hat{a}_{mn}^{u} = \frac{\partial E}{\partial u_{mn}} = 2 \sum_i \sum_j e_{ij} G_{ij}^{u} R_{ij}^{(mn)}$$  \hspace{1cm} (15)

$$\hat{a}_{mn}^{v} = \frac{\partial E}{\partial v_{mn}} = 2 \sum_i \sum_j e_{ij} G_{ij}^{v} R_{ij}^{(mn)}$$  \hspace{1cm} (16)

where:

$$R_{ij}^{(mn)} = \sum_k c_{mn} N^4(x_i - 2l + 2) N^4(y_j - 2k + 2) +$$

$$+ d_{mn} N^4(x_i - 2l + 3) N^4(y_j - 2k + 3) + g_{mn} N^4(x_i - 2l + 2)$$

$$N^4(y_j - 2k + 2) + h_{mn} N^4(x_i - 2l + 3) N^4(y_j - 2k + 2)$$

(17)

where $m$ and $n$ depend on $m, n, l, k$ and $-2 \leq m, n \leq 2$. The computation of the hessian matrix $H$ is needed requiring:

$$h_{mn}^{uu} = \frac{\partial^2 E}{\partial u_{mn} \partial u_{pq}} = 2 \sum_i \sum_j (G_{ij}^{u^2} R_{ij}^{(mn)} R_{ij}^{(pq)})$$  \hspace{1cm} (18)

$$h_{mn}^{vv} = \frac{\partial^2 E}{\partial v_{mn} \partial v_{pq}} = 2 \sum_i \sum_j (G_{ij}^{v^2} R_{ij}^{(mn)} R_{ij}^{(pq)})$$  \hspace{1cm} (19)

$$h_{mn}^{uv} = \frac{\partial^2 E}{\partial u_{mn} \partial v_{pq}} = 2 \sum_i \sum_j (G_{ij}^{u} G_{ij}^{v} R_{ij}^{(mn)} R_{ij}^{(pq)})$$  \hspace{1cm} (20)

The matrix $H$ is used to compute an updated flow vector by setting:

$$(H + \lambda D) \hat{\nabla} = \hat{\alpha}$$  \hspace{1cm} (21)

where $\lambda$ is a stabilization term [6], $D = \text{diag}(H)$, $\hat{\nabla}$ is an increment to the current displacement estimates $(\hat{u}_{mn}, \hat{v}_{mn})$ and $\hat{\alpha}$ is the vector of energy gradients $(\hat{a}_{mn}^{u}, \hat{a}_{mn}^{v})$. The approximation model adopted generates a matrix each entry involving the quantities:

$$R_{ij}^{(mn)} R_{ij}^{(pq)} \neq 0$$  \hspace{1cm} (22)

where:

$$x_i - l_1 \leq l_j + l_2 \leq x_i + 2$$

and:

$$y_j - l_1 \leq k_1 + k_2 \leq y_j + 2$$

and the indexes $l_j, k_j$ regard the sums in $R_{ij}^{(mn)}$ while $l_2, k_2$ regard the sums in $R_{ij}^{(pq)}$. A linear system with the same structure of the previous system (21) must be solved at each
step of the Levenberg-Marquardt algorithm. As already underlined in section 2 the quasi-interpolant operator adopted, combined with a scaling process, generates a coarse-to-fine process and at each level the described algorithm must be applied.

4 Parallel Issues
This section presents the parallel features of the model previously described by supposing to work with a multiprocessor system equipped with NP processors. The parallelism has been exploited by involving the geometric parallel paradigm.

4.1 Data distribution
A parallel algorithm is penalized from the communication overhead mainly due to the data distribution, data exchange and collection of local results. Therefore, it is important to determine the best data partitioning by considering that the solving scheme involves local operators requiring information from neighbour values for each application point. When a processor needs to compute operations on boundary elements, the local operator will also require values relative to points belonging to adjacent processors. Consequently, a communication overhead that depends on the length of the boundaries will occur besides the initial data distribution.

In the following two data partitions, denoted as block and strip partition, have been taken into account.

a) Block partition
The data are divided into blocks amongst the NP processors arranged along a 2-dimensional mesh of size \( P_x \times P_y \). Let \( s \cdot t \) the number of points on a generic grid and \( s_p \cdot t_p \) those belonging to a generic processor where \( s_p = \lfloor s/P_x \rfloor \), \( t_p = \lfloor t/P_y \rfloor \).

When local operators are applied during the computational process, each processor will be involved in 2 data transfer operations with each of the neighbouring processors (fig.2). The amount of data exchanged by each processor is \( 4h(s_p+t_p) \) at most where \( h \) is the width of set values required by the operator in use.

The total data transfer operations will be \( 4(2NP-P_x-P_y) \), whilst the total amount of data to be transferred will be \( 4h(s+1)NP - (s+t) \).

b) Strip partition
Using this partition, the data are divided into strips amongst a 1-dimensional mesh of NP processors (fig.3). In this case, each processor will be involved at most in 4 data transfer operations. Hence, the total data transfer will be \( 4(NP-1) \) and by indicating with \( a \) the lowest value between \( s \) and \( t \), the total amount of data to be exchanged will be \( 4a(NP-1) \).

Therefore, the best strip partition will be carried out going on along the largest dimension. Working with images almost squared, i.e. \( s = t \), the block partition involves a less amount of data transfer than the strip one but needs more transfers. These considerations are relevant for distributed architectures where it is preferable to minimize the number of communications among processors.

4.2 Computations and communications
The input data \( I_0 \) and \( I_1 \) are involved in a different manner in the solving scheme. \( I_0 \) is not modified during the overall process and it can be partitioned at the beginning of the computation among all the available processors. The values of the image brightness \( I_1 \) are interested in numerical derivatives computed with difference schemes and must be updated. Hence, a suitable data exchange must be performed for \( I_1 \) (fig.2-3) between adjacent processors.

By considering the strip data partition for instance along the horizontal direction, let \( m^{(0)} \cdot n^{(0)} \) be the initial velocity guesses number on each processor. As reported in section 2, the quasi-interpolant is a local operator that for each point requires the close values of the velocity vector components.
The velocity components are partitioned among the processors following the same rule used for $I_l$ but with a larger amount of data to be exchanged. In fact, for each value $u(i,j)$ to be computed, a processor uses the values in the set $\{ \hat{u}(l_i,j_j) \mid i-1 \leq l_i \leq i+2, j-1 \leq l_j \leq j+2 \}$.

Analogous considerations are extended to the $v(i,j)$ computations. For each communication the total amount of transferred data for computing $u^{(k)}_{ij}$ and $v^{(k)}_{ij}$ at a generic level $k$ is upper bounded by $2^{(k+3)}m^{(0)}$. The quasi-interpolant operator is applied a number of times equal to $2^{(k+3)}3(m^{(0)-1})(m^{(0)}-1)+m^{(0)}+n^{(0)}+2$.

Now, let $m^{(0)}$ and $n^{(0)}$ be the initial number of data on each processor in which the numerical derivatives $I^{x}_{i}$, $I^{y}_{i}$ must be computed. Data transfers are required only for the $I^{x}_{i}$ derivatives, i.e. each processor performs at most $4m^{(0)}$ exchanges regarding the boundary values. Then at the level $k$, each data transfer involves $2^{(k+3)}(m^{(0)}-1)+l$ values and the total number of derivatives which must be computed are $2^{(k+3)}(m^{(0)}n^{(0)}+1)+2$.

Similar operative modalities must be performed for computing the derivatives $G_{ij}^{x}$ and $G_{ij}^{y}$ subsequently involved in the computational scheme.

The values obtained up to now allow to build a linear system. Each processor can compute the known vector and the matrix entries regarding velocity values belonging to itself. To this end, it is necessary to exchange data regarding the $G_{ij}^{x}$, $G_{ij}^{y}$ and $e_{ij}$ evaluated at the boundaries. The amount of data involved for each data transfer is the same as the one required by the quasi-interpolant operator. Each processor can compute the $P_{ij}^{(m,n)}$ for its $(m,n)$ points without any data transfer. Therefore, the known vector and the hessian matrix are already distributed among all processors. The hessian matrix is a sparse multibanded matrix and by applying a suitable $\hat{a}$ parameter, an iterative solver can be successfully employed. In order to compute the matrix-vector products which are involved at each step of the iterative solver, only few values of the updated vector must be exchanged between adjacent processors.

As a conclusion, the computational process is suitable to run on distributed multiprocessors platforms resulting synchronous, with a good tasks balancing and requiring an exiguous amount of data transfer.

References: