COMPUTATING OPTICAL FLOW USING PIPELINE ARCHITECTURE

S. Negahdaripour

Department of ECE
University of Miami
USA

A. Shokrollahi

InterSystems Corporation
USA

Abstract Accurate estimation of motion from time-varying imagery has been a popular problem in vision studies. This information can be used in segmentation, 3D motion and shape recovery, target tracking, and other problems in scene analysis and interpretation. We have presented a dynamic image model for estimating image motion from image sequences, and have shown how the solution can be obtained from a set of partial differential equations. In this paper, we have investigated a relaxation type algorithm for obtaining a numerical solution to these equations, and considered the implementation of the algorithm on a variation of the general pipeline interconnection scheme using transputers. This architecture is compared against two others based on flexibility and efficiency. It is observed that with respect to computation, a mesh connected architecture has advantages over the proposed pipeline scheme. However, the pipeline configuration is easily expandable and more robust to changes in the algorithms parameters and image size.

Key Words Vision, Image Motion, Optical Flow, Pipeline, Mesh

چکیده تخمین دقیق حرکت از طریق تصویربرداری در مقاطع مختلف زمانی، مسئله مهمی در مطالعات بینایی بوده است. موارد استفاده از این اطلاعات در زمینه های قطعه بندی، حرکت سه بعدی و بازیابی شکل، به دنبال هدف گشتن و مسائل دیگر تحلیل و درک صحنه بوده است. ما مدل پویایی را برای تخمین حرکت تصویر از روی سری تصویرها ارائه نموده ایم و نشان داده ایم که چگونه راه حل مسئله می تواند از طریق یک سری معادلات دیفرانسیل پاره ای بدست آید. در این مقاله یک الگوریتم نوع استراحت برای دستیابی به راه حلی عددی برای این معادلات مورد بررسی قرار گرفته و همچنین پیاده سازی این الگوریتم در محیط اتصال خط لوله عمومی با استفاده از ترانسپیوتر ارائه خواهد شد. این معماری، بر اساس انعطاف پذیری و بازدهی، با دو مورد دیگر مقایسه خواهد شد. نتیجه بدست آمده نشان می دهد که از نظر محاسباتی، معماری شبکهٔ مربعی، مزایایی نسبت به روش خط لوله ای دارد. با این حال، سیستم خط لوله ای براحتی قابل گسترش می باشد و در برابر تغییرات در پارامترهای الگوریتم و اندازه تصویر انعطاف پذیری بیشتری دارد.

INTRODUCTION

Vision is the most powerful sensing capability of humans and most other biological systems. In over two decades, computer vision researchers have been involved in research problems related to giving machines "Some sense of visual perception" in order to function in their environment with intelligence. Until about a decade ago, the most significant bottleneck in the realization of such systems was lack of the technology to meet the computational requirements [1]. Because of the amount of data involved, even today's conventional single-CPU serial computers are unable to provide sufficient processing power for real-time performance in most vision problems. Therefore, the need for special purpose multiprocessor hardware is immanent. An important question is how to arrange the processors

in an interconnection scheme which would achieve the highest efficiency and speed up possible. In [2], the authors discuss a number of mesh connected architectures for image processing and vision. In their work, they develop efficient algorithms to support data routing among the processors. Pipeline architectures have proven to be effective for image processing [3], but are seldom used for vision tasks. With the introduction of transputers to the world of computing, the task of interconnecting processors has become easier. Transputers are processing elements (processor and memory) with four built-in communication channels. They are designed to be easily connected in mesh fashion and to communicate with each other over their communication channels. This is a particularly useful feature of transputers for vision applications.

Vision problems involve processing two-dimensional images to extract useful information about a three-dimensional world and the objects in it. Useful information refers to what permits an autonomous agent or robot to interact with its surrounding environment intelligently. For example, a robot needs to detect nearby objects and avoid them in order to navigate without collision in rooms, hallways, or outdoors. To do so, a robot equipped with optical sensors should be able to determine its distance to obstacles as well as its motion relative to stationary or moving objects using visual cues.

The change in the positions of objects in a scene relative to the camera induces variations in image brightness patterns. The perceived motion of these patterns, referred to as *optical flow*, contains rich information about the scene. It can be used to infer an object's shape, position, spatial arrangement, and/or motion relative to the camera. Efficient computation of optical flow with an acceptable level of accuracy has been one of the most important research topics in vision. The root of the problem dates back to the psychophysics studies on human visual perception in the early part of the century [4]. Of various types of approaches to solving this problem (e.g., see [5] for a survey), *gradient-based methods* have particularly been attractive for obtaining a dense, yet reasonably accurate, estimate over the whole image, pioneered by the work in [6].

In this paper, we have investigated the implementation of a method for the computation of optical flow [7]. The solution to the problem is given by a set of partial differential equations, based on a formulation in variational calculus, which can be determined numerically using a relaxation type algorithm. We have considered issues related to the implementation of the algorithm on a variation of the general pipeline interconnection scheme using transputers. This architecture is compared against two others based on flexibility and efficiency. It is shown that with respect to computation, a mesh connected architecture has advantages over the proposed pipeline scheme. However, the pipeline configuration is easily expandable and more robust with respect to changes in the algorithms parameters and image size.

PRELIMINARIES

Let $e(\mathrm{r})$ denote the image brightness of some scene point, where $\mathrm{r} = [x,y,t]^T$ is a point in the image-time volume. At some later time $t+\delta t$, the scene point may move relative to the camera and thus projects onto a new point $\mathrm{r} + \delta\mathrm{r} = [x + u\delta t, y + v\delta t, t+\delta t]^T$. In addition, the image brightness of the point can vary by $\delta e = e(\mathrm{r} + \delta\mathrm{r}) - e(\mathrm{r})$. Computation of *image motion/flow* involves determining from an image sequence, the displacement $[u,v]^T$ during the time interval t to $t + \delta t$ for every image point r.

Negahdaripour et al. [7, 8] have investigated the application of the constraint equation

$$E_t + E_x u + E_y v - Em_t - c_t = 0 \qquad (1)$$

for the computation of optical flow. This equation is derived from the dynamic image model

$$E(r + \delta r) = M(r)\,E(r) + C(r), \quad (2)$$

which permits variations in the brightness of an image point according to a linear transformation model, involving multiplier and offset fields M and C. We have used $\delta m = M - 1$ and $\delta c = C - 0$ to represent the transformation fields M and C in terms of variations about 1 and 0, respectively. The difficulty in solving the optical flow problem is that, at each image point, four unknowns (u, v, m_t, and c_t) need to be determined. However, Equation 1 provides only one constraint on the sought after unknowns.

In overcoming this problem, we note that the four fields vary smoothly over most regions of the image. Discontinuities in depth (for example, at occluding boundaries) give rise to discontinuities in the optical flow. Also, object motions may be different across occluding boundaries, which also give rise to discontinuities in the optical flow. Additionally, discontinuities can be expected in m_t and c_t, if illumination conditions or reflection properties that depend on surface material change abruptly as the surface moves in the envirnoment. However, these discontinuities are typically restricted to isolated regions in the image. Based on these facts, it is a reasonable assumption to require that the optical flow and the transformation fields should vary smoothly from one image point to the next.

One way to impose smoothness on some field f is to minimize the integral of its gradient magnitude over the whole image:

$$e_f = \iint (f_x^2 + f_y^2)\,dx\,dy, \quad (3)$$

where $f_x = \partial f/\partial x$ and $f_y = \partial f/\partial y$. A measure of departure from smoothness according to Equation 3 can be written for each of the related fields u, v, m_t, and c_t, and combined with an appropriate weighing factor:

$$e_s(u,v,m_t,c_t) = \lambda_u e_u + \lambda_v e_v + \lambda_m e_m + \lambda_c e_c, \quad (4)$$

where the constants λ_u, λ_v, λ_m, and λ_c are the weights for each term.

Finally, the problem can be formulated as a minimization problem by combining the error in the optical flow constraint equation and the smoothness measure into a single functional formula:

$$e(u,v,m_t,c_t) = e_s(u,v,m_t,c_t) + \iint (E_t + E_x u + E_y v - E m_t - c_t)^2\,dxdy \quad (5)$$

Minimizing e by the appropriate choice of the four fields, u, v, m_t, and c_t is a problem in variational calculus. It involves the application of the Euler-Lagrange equation,

$$\frac{\partial \phi}{\partial f} - \frac{\partial}{\partial x}\left(\frac{\partial \phi}{\partial f_x}\right) - \frac{\partial}{\partial y}\left(\frac{\partial \phi}{\partial f_y}\right) = 0, \quad (6)$$

where ϕ is the integrand in the cost functional e, and f is each of u, v, m_t, or c_t. We have shown in Negahdaripour et al. [7] that the discrete implementation of the resulting equations can be written in the form $x = A^{-1} g(\bar{x})$, where

$$x = \begin{bmatrix} u \\ v \\ m_t \\ c_t \end{bmatrix}, \quad g(\bar{x}) = \begin{bmatrix} \lambda_u \bar{u} - E_x E_t \\ \lambda_v \bar{v} - E_y E_t \\ \lambda_m \bar{m_t} + E E_t \\ \lambda_c \bar{c_t} + E_t \end{bmatrix}$$

$$A = \begin{bmatrix} E_x^2 + \lambda_u & E_y E_y & -E_x E & -E_x \\ E_y E_y & E_y^2 + \lambda_u & -E_y E & -E_y \\ -E_x E & -E_y E & E^2 + \lambda_m & E \\ -E_x & -E_y E & E & 1 + \lambda_c \end{bmatrix}$$

In these equations,

$$\bar{f} = \frac{1}{4}(f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)) \quad (7)$$

represents the average of each field f over the four (north, south, west, and east) neighbors of an image point. Since the unknowns at each image point depend on the average of the fields over the neighboring cells, the solution to this problem is typically obtained from an iterative scheme of the form

$$x^{k+1} = A^{-1} g(\bar{x}^k), \quad (8)$$

where k is the iteration index, This iterative scheme, sometimes referred to as *Jacobi iteration*, requires updating the value at each image position at iteration $k+1$ based on the average values of the 4 neighbors from estimates at iteration k. The number of iterations needed to arrive at an accurate solution depends on the initial guess. It is desirable to have an initial guess which is a good estimate of the solution. For a sequence of images taken at a rate of 30 frames per second and assuming a smooth motion, the flow computed for some past frame (ideally last frame) is generally a good initial guess of the flow in the current frame. In most cases, only a few iterations are therefore sufficient to obtain a reasonably accurate solution. In experiments with 64×64 images of various texture levels, including the example in this paper, about 10 iterations, starting with an initial guess from previous frames, has been sufficient for convergence [9]. Needless to say, if the motion changes rapidly between frames, many more iterations may be required before an acceptable solution is obtained. Efficient multi-resolution schemes to address such problems have been proposed [10].

These types of algorithms adapt well to parallel implementation. The approach taken in this paper is that of domain decomposition. In domain decomposition, a set of data is divided among the computational nodes of the parallel processor system (another example of this approach is that of numerical integration). In the sections that follow, we discuss three interconnection schemes and point out advantages and disadvantages of each.

First, a squared mesh-connected architecture is described. Here, the information is loaded into the processors before the computation is started. Next, a general pipeline is discussed, where operations are performed on streams of data. Finally, a variation of the general pipeline interconnection is suggested. This interconnection attempts to address the shortcomings of the general pipeline architecture. These architectures are compared based on the speed, hardware complexity, and flexibility.

SPECIAL ARCHITECTURES

Since the solution of the algorithm is obtained from the Jacobi iteration, the need for processor communication is immanent. Although any processing element with four communication channels can be used, we have chosen transputers (IMS T801) as an example. The IMS T801 also includes a 64 bit floating point unit that is rated at 4.3 Mflops peak at 30 MHz. Experiments with various multiprocessors and multicomputers have shown that a processor rated at 4.3 Mflops will deliver about 1 Mflops for our type of computations.

DESIGN ISSUES

In order to design a special purpose architecture for implementation of the optical flow algorithm, two important issues should be considered. One is the computation power needed to carry out the implementation in real time and the other is the communication pattern among the processors. The number of processors needed is directly related to the amount of computation done and inversely proportional to the Mflop rating of the processor. The communication pattern among the processors is

dictated by the nature of the algorithm and the choice of partitions.

Computational Issues

To calculate the required computational power for real-time processing, the algorithm was coded on a machine with a known Mflop rating and execution time of the program was measured. Every part of the calculation was timed and using the Mflops observed, the number of floating point operations needed for every part was calculated. These parts consist of calculating necessary image gradients, constructing A^{-1}, and iterating once.

Communication Issues

A transputer messaging time is calculated using $2\times W+19$ cycles (33 nano seconds per cycle). In this fomula W is the number of words in a message (4 bytes per word). The communication time among transputers depends not only on the number of words in each message, but also on the number of messages sent. Therefore, it is desirable to minimize the product of the above quantity and the number of messages sent.

INTERCONNECTION SCHEMES

Mesh

Using the estimated transputer rating discussed earlier, it is calculated that about 64 transputers are needed to compute optical flow in real time (30 frames/second). The communication among the transputers is achieved through all four channels (Figure 1). Since transputers contain only 4 Kbytes of local memory, 65536 bytes of external memory is needed for a 64×64 image. Four high speed buses (minimum bandwidth of 65536 bytes/sec) connect the links on the four sides of the mesh. Two of these buses are used to load the image and unload the results from the transputers and the other two are for general purposes.

The computation of optical flow for an image sequence is performed as follows: A 64×64 image, sampled at one frame per thirtieth of a second, is loaded through the top transputers. Each transputer holds an 8×8 patch of the image. Each transputer calculates image gradients within the patch and constructs A^{-1}. It then performs 10 iterations, each time communicating an array of eigtht float numbers containing the results (u, v, m, and c) for the border pixels to each of its neighbors. This is a cost of $(2\times 4\times 8+19)=83$ cycles each time the array is sent or received. This is sent to and received from each of the 4 neighbors. Thus, the total messaging time is $(8\times 83)\times\frac{1}{30\,MHz}=0.0221$ msec. Using the measured times for the computation of gradients, construction of A^{-1}, and 10 iterations, the total processing time is $(0.01318\times 64+0.04963\times 64+10\times 0.03903\times 64)=29.0182$ msec for 64 pixels. To unload the results, each bottom-row transputer takes turn putting its results and those from the transputers above it on the bus. Meanwhile, the top transputers get the next

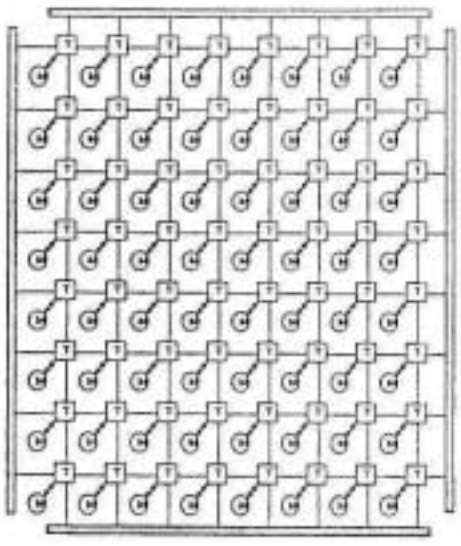

Figure 1. Conventional mesh architecture with high-speed buses connecting free links.

frame of the sequence and communicate it down. The unload time for each packet is $(2 \times 4 \times 64 + 19) \times 1/30MHz = 0.0177$ *msec*, and the total time for transferring 64 packets is $64 \times 0.0177 = 1.1328$ *msec*. The total communication and computation time amounts to 0.0221 + 29.0182 + 1.1328 = 30.1731 *msec*, which is within one thirtieth of a second.

The proposed mesh architecture performs well for the particular image size and number of iterations; the mapping of data is straightforward, the communication bandwidth is low, and real time performance is achieved. However, some applications may require more iterations for convergence, depending on the quality of data and the type and amount of motion. This architecture cannot be easily modified to accommodate more iterations, while maintaining the same topology.

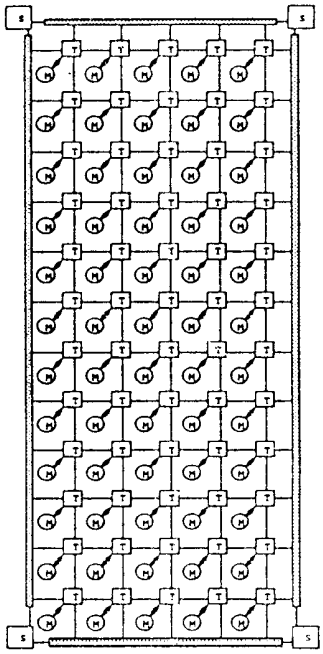

Figure 2. Conventional transputer pipeline with switches connecting four high-speed buses.

Pipeline

Based on computation time measurements, it is calculated that 5 parallel pipes of 10 stages each are needed for 10 iterations in real time. Additionally, 2 transputers are needed for computing gradients and constructing A^{-1} for each of the 5 pipes. This amounts to 12 stages per pipe as shown in Figure 2. Four high-speed buses (minimum bandwidth of 65536 bytes/sec) connect the free links on the sides. Each pair of adjacent buses are connected through a switch box. Therefore, contents of each bus can be linked onto any other bus.

When performing the computation on the mesh connected architecture, the final solution for the previous frame is already resident in the memory, which can be used as the initial guess for the current frame. This is not the case with a pipeline architecture. The available solution, when the first pipe stage starts its computations, is the solution of the last pipe stage from 10 frames before (10 is the depth of the pipe). If the motion is reasonably smooth in time, this solution is a good initial guess. In the implementation discussed here, this is assumed to be the case. (Examples supporting this assumption are presented in [9].)

The first stage (row) of the first pipe (column) takes the initial guess of the solution and the first 819 (64 × 64 pixels/5 pipes) column-major pixel values, the first stage of the second pipe takes the next 819 column-major pixel values and initial guesses, and so on. Half of the information is sent down to the second stage (row). These two stages compute the gradients and A^{-1} for the whole image. The A^{-1} data along with the initial guess (unprocessed) is passed on to the third stage (row) to start the iterative calculation of optical flow. Meanwhile, the first two stages get the next frame and the initial guess. From here, each stage communicates its border values and performs one iteration. The results of this iteration and the A^{-1} matrix are then sent down the pipe to the next stage and the results for the next frame is received from the previous stage. This procedure is repeated at every stage until the final result (from the last row) is ready to be put on the output bus and also rerouted to the

input bus through one of the side buses.

Between the stages, one needs to communicate u, v, m, and c as well as 10 elements of A^{-1} (the matrix is symmetric). The time for communicating between stages is $2 \times (2 \times 14 \times 819 + 19) \times 1/30\,MHz = 1.5300$ *msec*. Receiving data from the top bus (image and initial guess takes $((2 \times 1024 + 19) + (2 \times 4 \times 819 + 19)) \times 1/30\ MHz = 1.0967$ *msec*, but this occurs during the 1.5300 *msec* time interval for the communication between the stages. Computing one iteration for 819 pixels takes $0.03906 \times 819 \times 30 = 31.9922$ *msec*. The communication of border information consists of sending and receiving a message to and from the neighbors on each side. A column of results (4 computed fields for 64 pixles) is communicated. This takes $2 \times 2 \times (2 \times 4 \times 64 + 19) \times 1/30\,MHz = 0.0708$ *msec*. The total communication and computation time is $0.0708 + 1.5300 + 31.9922 = 33.5930$ *msec*, which is just over one thirtieth of a second. An advantage of the pipeline is that it is easily reconfigurable to accommodate different number of iterations. This and/or further processing of the results can be accomplished by adding extra stages to the pipe without violating the design principles and topology. The shortcoming of the pipe, compared to the mesh, is limitation on motion variations per frame. This is due to the fact that the initial guess for each frame is the solution from 10 frames before. Additionally, it requires a relatively large body of data to be communicated between various stages. This problem can be solved by using semi-shared memory between every two stages of the pipe, as we discuss next.

Modified Pipeline

This architecture is similar to the previous one with one fundamental difference; there are no links between the different stages of the pipe, except for the first two stages. The portion of the image needed in the second stage is passed on through this link. In this design, each transputer T_k involved in iteration k has access to four external memory modules, two for reading (M_k^{r1} and M_k^{r2}) and two for writing ($M_k^{\omega 1}$ and $M_k^{\omega 2}$)(see Figure 3). Note that $M_k^{\omega 1}$ and $M_k^{\omega 2}$ are the same as M_{k+1}^{r1} and M_{k+1}^{r2} in this notation. Transputer T_k reads the results of iteration k - 1 on image n from $M_{k-1}^{\omega 1}$ (M_k^{r1}), performs one iteration, and writes the output to $M_k^{\omega 2}$ (M_{k+1}^{r2}). Meanwhile, transputer T_{k+1} reads the results of iteration k on image n-1 from $M_k^{\omega 1}$ (M_{k+1}^{r1}), performs one iteration, and writes the output to $M_{k+1}^{\omega 2}$. At the next time interval, the access to memory modules reverses. Transputer T_k reads the results of iteration k-1 on image n+1 from $M_{k-1}^{\omega 2}$ (M_k^{r2}), performs one iteration, and writes the output to $M_k^{\omega 1}$ (M_{k+1}^{r1}). Meanwhile, transputer T_k+1 reads the results of iteration k on image n from $M_k^{\omega 2}$, performs one interation, and writes the output to $M_{k+1}^{\omega 1}$ (see Figure 4).

The computation of gradients and A^{-1} on 10

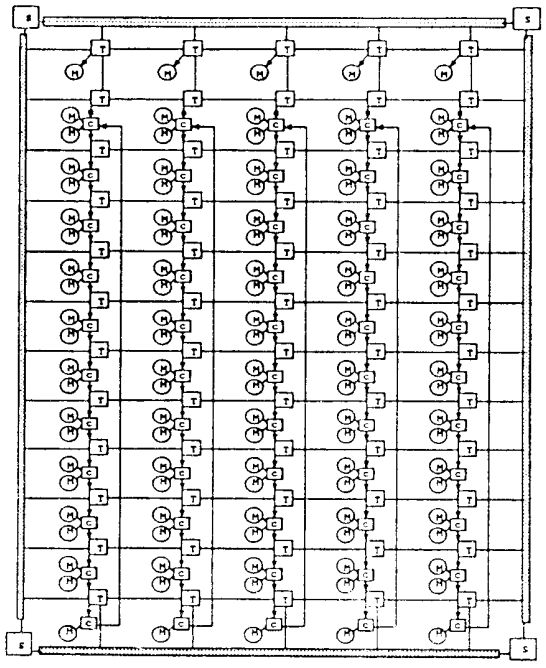

Figure 3. Modified transputer pipeline with memory modules substituting the links.

transputers is 25.75 *msec* and receiving and sending image portions takes about 0.07567 *msec*. Sending A^{-1} to the second stage takes 0.2733 *msec*. Therefore, the results are ready for the third stage within approximately 26.1033 *msec*. Each iteration takes 31.9923 *msec* and sending the results on the bus via links for 5 transputers taking turns takes $(2 \times 4 \times 819 + 19) \times 5 \times 1/30\ MHz = 1.09666\ msec$. In addition, communicating the results at each iteration takes 0.0708 *msec*. Thus, the total computation and communication time is about 33.1597 *msec* (within frame rate).

Using semi-shared memory modules, the communication among the different stages of the pipe is eliminated. This architecture, as in the previous case, is easily reconfigurable to allow more or fewer iterations. This is achieved through duplication or removal of stages of the pipe. Also, the feedback connection providing initial guess can be placed at any stage.

The feedback lag of the initial guess is directly proportional to the depth of the pipe. The architecture can be reconfigured to 10 parallel pipes of 6 stages each, instead of 5 pipes of 12 stages. The first stage performs the gradient calculations and froms A^{-1}. The remaining stages perform 2 iterations on each data set. In this architecture, there would be no links between the stages and all transputers will have semi-shared memory. By doing so (reducing the depth) one can feed back the results from 5 frames before as the initial guess for the new frame. If one chooses to perform one more iteration, another stage needs to be added to the pipe. This costs 10 transputers for the 6-stage pipe as opposed to 5 transputers for the 12-stage pipe. If two more iterations are needed, both architectures result in the same cost (10 transputers). By lowering the depth of the pipe and expanding the width, one gains flexibility with respect to the motion sizes that the pipe can handle, but looses efficiency in expanding the pipe at low cost. Different depths can be achieved in the same manner.

Additional pipes can be added to the system by placing them in parallel with the existing ones. Doing so enables processing of 819 more pixels per pipe added. In the case where the pipe is only six stages deep, addition of another pipe will result in the ability to process 410 more pixels. Of course, this only requires half as many transputers.

AN EXPERIMENT

A sequence of 75 images were taken, simulating the motion of an underwater vehicle near an ocean bottom-floor (see Figure 5a). The scene is a sheepskin curtain, shown in Figure 5b, with similar surface texture properties as that of the ocean floor. The optical flow

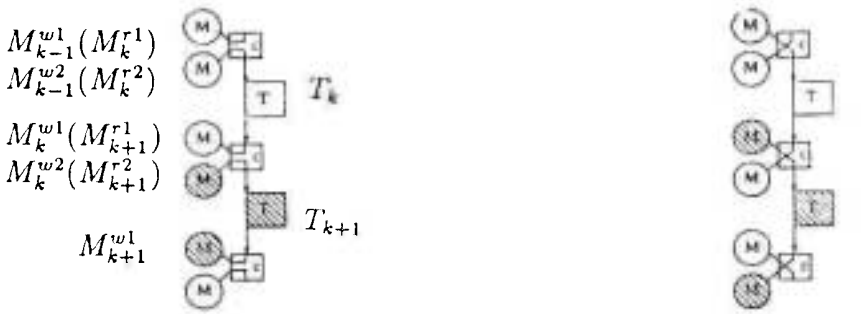

Figure 4. Memory access method, with configurations **(a)** at time t and **(b)** at time $t + \delta t$

for each frame k was computed from 10 iterations of the algorithm, using as initial guess the solution from frame k-10. Only the flow for the last frame is shown in Figure 5c due to space limitations. The flow shows that the vehicle motion is towards the scene and to the right. From the optical flow, the motion of the vehicle can be computed using a number of techniques. Using the method in [11], we estimated the vehicle translation between each frame. The error in the computed motion-vs-iteration number for the last sequence, given in Figure 5d, shows the convergence within a few percent of the true solution in about 10 iterations.

SUMMARY

We have investigated iterative computation of optical flow for image sequences using 3 interconnection schemes. An 8×8 mesh has been found sufficient for real-time computation on 64×64 images, permitting 10 iterations per frame using as the initial guess the solution from the previous

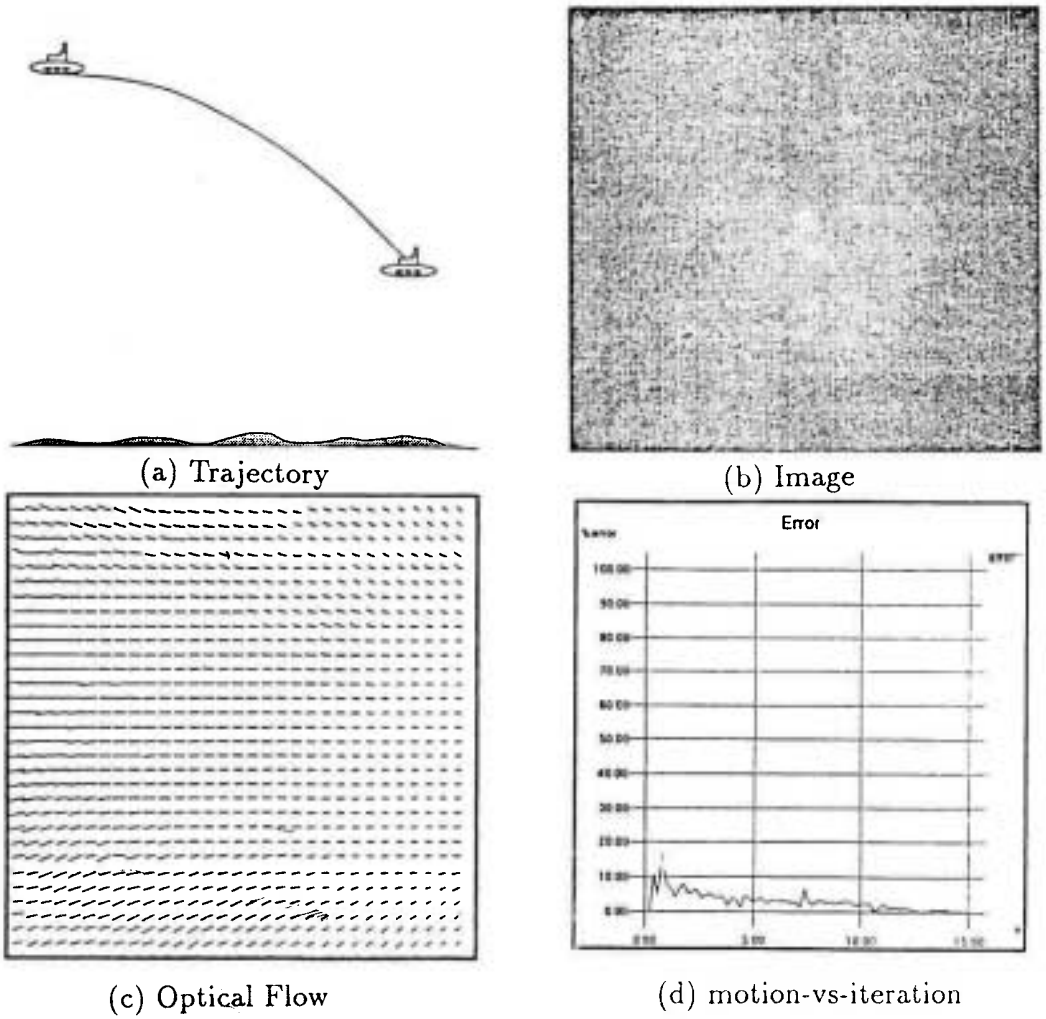

Figure 5. A Selected experiment showing motion trajectory (**a**) a particular image of the low-textured scene (**b**) estimated optical flow for the last frame and (**c**) error in the estimate motion-vs-frame number.

frame. However, the reconfiguration to accommodate variations in image size and/or the number of iterations can not be done efficiently. A general pipeline architecture overcomes this problem, but requires an extensive amount of communication. A modified pipeline scheme has been proposed to eliminate the need for communication between stages of the pipe. As in the original pipeline, it can be easily expanded by duplicating its stages. In the pipeline architecture, the initial guess of the iterative process on image n is the solution from image n-k, where k is the depth of the pipe. This imposes some limitations on the acceptable motion variations between consecutive frames. The modified pipeline permits feeding the information from any intermediate stage back to the first stage. Both architectures allow extending the width of the pipe and reducing its depth, thus decreasing the lag in the initial guess feedback. This is achieved at the expense of less efficiency, since each stage of the pipeline gets less of the image to process. However, this permits more iteratons to be performed, if necessary.

ACKNOWLEGEMENT

Funding for the research was provided in part by the NSF under grant IRI-8910967.

REFERENCES

1. S. Negahdaripour and A. K. Jain, "Final Report of the NSF Workshop on Grand Challenges in Computer Vision: Future Research Direction in Computer Vision," (November 1991).
2. V. K. Prasanna-Kumar and D. Reisis, "Parallel Architectures for Image Processing and Vision," Proc. DARPA Image Understanding Workshop, (April 1988).
3. R. P. W. Duin and P. P. Jonker, "Processor Arrays Compared to Pipelines for Cellular Image Operations," Multicomputer Vision, S. Levidaldi (ed.), Academic Press, (1988).
4. H. Van Helmholtz, "Helmholtz's Traetsie on Physiological Optics, " *Journal Optical Society of America*, J. P. Southall (ed), (1925).
5. J. L. Baron, D. J. Fleet, S. S. Beauchemin and T. A. Burkitt, "Performance of Optical Flow Techniques," TR-299, Dept. Comp. Science, University of Western Ontario, (March 1993).
6. B. K. P. Horn and B. G. Schunck, "Determining Optical Flow," *Artificial Intelligence*, Vol. 17, (1981).
7. S. Negahdaripour, A. Shokrollahi and M. Gennert, "Relaxing the Brightness Constancy Assumption in Computing Optical Flow," Proc. Int. Conf. Image Processing, Singapore, (September 1989).
8. S. Negahdaripour and C. H. Yu, "A Generalized Brightness Change Model for Computing Optical Flow," Proc. 4th International Conference on Computer Vision, Berlin, Germany, (May 1993).
9. A. Shokrollahi, "Computing Optical Flow: A Parallel Implementation for Transputers," M. S. Thesis, Department of Electrical Engineering, University of Hawaii, (December 1990).
10. D. Terzopoulos, "Image Analysis Using Multigrid Relaxation Methods," *IEEE Trans. PAMI*, Vol. 8, No 2, (March 1986).
11. A. R. Bruss and B. K. P. Horn "Passive Navigation," *CVIGIP*, Vol. 21, No 1, (January 1983).