October 1989

NAG1-613

UILU-ENG-89-2234
CSG-111

LANGLEY
GRANT
IN-61-CR
23403
458

COORDINATED SCIENCE LABORATORY
College of Engineering

NOVEL TECHNIQUES FOR DATA DECOMPOSITION AND LOAD BALANCING FOR PARALLEL PROCESSING OF VISION SYSTEMS: IMPLEMENTATION AND EVALUATION USING A MOTION ESTIMATION SYSTEM

Alok N. Choudhary
Mun K. Leung
Thomas S. Huang
Janak H. Patel

(NASA-CR-185991) NOVEL TECHNIQUES FOR DATA DECOMPOSITION AND LOAD BALANCING FOR PARALLEL PROCESSING OF VISION SYSTEMS: IMPLEMENTATION AND EVALUATION USING A MOTION ESTIMATION SYSTEM (Illinois Univ.) 45 p

N90-12414

Unclas

G3/62 0233263

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Approved for Public Release. Distribution Unlimited.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-89-2234 (CSG-111)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA / NSF
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) NASA Langley Research Center Hampton, VA 23665 National Science Foundation 1800 G Street, NW / Washington DC 20550
8a. NAME OF FUNDING / SPONSORING ORGANIZATION NASA / NSF 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA 1-613 NSF IRI 87-05400
8c. ADDRESS (City, State, and ZIP Code) see 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Novel Techniques for Data Decomposition and Load Balancing for Parallel Processing of Vision System: Implementation and Evaluation using a Motion Estimation System

12. PERSONAL AUTHOR(S)
Alok N. Choudhary, Mun K. Leung, Thomas S. Huang and Janak H. Patel

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	89/10	43

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	parallel processing, vision, motion, stereo load balancing, hypercube

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Computer vision systems employ a sequence of vision algorithms in which the output of an algorithm is the input of the next algorithm in the sequence. Algorithms that constitute such systems exhibit vastly different computational characteristics, and therefore, require different data decomposition techniques and efficient load balancing techniques for parallel implementation. However, since the input data for a task is produced as the output data of the previous task, this information can be exploited to perform knowledge based data decomposition and load balancing.

First, this paper presents algorithms for a motion estimation system. The motion estimation is based on the point correspondence between the involved images which are a sequence of stereo image pairs. We propose algorithms to obtain point correspondences by matching feature points among stereo image pairs at any two consecutive time instants. Furthermore, the proposed algorithms employ non-iterative procedures, which results in saving considerable amount of computation time. The system consists of the following steps: 1) extraction of features, 2) stereo match of images in one time instant, 3) time match of images from

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

consecutive time instants, 4) stereo match to compute final unambiguous points and, 5) computation of motion parameters. Second, this paper presents several techniques to perform static and dynamic load balancing for computer vision system applications. These techniques are novel because they capture the computational requirements of a task by examining the data when it is produced. Furthermore, these techniques can be applied to many vision systems because a great deal of algorithms in different systems are either same, or have similar computational characteristics. Finally, these techniques are evaluated by applying them to a parallel implementation of the algorithms of the motion estimation system. The main issues considered in parallel implementation are utilization of processors, communication among processors, and load balancing. It is shown that the performance gains when these data decomposition and load balancing techniques are used, are significant, and the overhead of using these techniques is minimal. The performance is evaluated by implementing the algorithms on a hypercube multiprocessor system.

Novel Techniques for Data Decomposition and Load Balancing for Parallel Processing of Vision Systems: Implementation and Evaluation using a Motion Estimation System

Alok N. Choudhary, Mun K. Leung, Thomas S. Huang and Janak H. Patel

Coordinated Science Laboratory
University of Illinois
1101 W. Springfield
Urbana, IL 61801

Abstract

Computer vision systems employ a sequence of vision algorithms in which the output of an algorithm is the input of the next algorithm in the sequence. Algorithms that constitute such systems exhibit vastly different computational characteristics, and therefore, require different data decomposition techniques and efficient load balancing techniques for parallel implementation. However, since the input data for a task is produced as the output data of the previous task, this information can be exploited to perform knowledge based data decomposition and load balancing.

First, this paper presents algorithms for a motion estimation system. The motion estimation is based on the point correspondence between the involved images which are a sequence of stereo image pairs. We propose algorithms to obtain point correspondences by matching feature points among stereo image pairs at any two consecutive time instants. Furthermore, the proposed algorithms employ non-iterative procedures, which results in saving considerable amount of computation time. The system consists of the following steps: 1) extraction of features, 2) stereo match of images in one time instant, 3) time match of images from consecutive time instants, 4) stereo match to compute final unambiguous points and, 5) computation of motion parameters. Second, this paper presents several techniques to perform static and dynamic load balancing for computer vision system applications. These techniques are novel because they capture the computational requirements of a task by examining the data when it is produced. Furthermore, these techniques can be applied to many vision systems because a great deal of algorithms in different systems are either same, or have similar computational characteristics. Finally, these techniques are evaluated by applying them to a parallel implementation of the algorithms of the motion estimation system. The main issues considered in parallel implementation are utilization of processors, communication among processors, and load balancing. It is shown that the performance gains when these data decomposition and load balancing techniques are used, are significant, and the overhead of using these techniques is minimal. The performance is evaluated by implementing the algorithms on a hypercube multiprocessor system.

1. Introduction

Computer vision tasks employ a broad range of algorithms. In vision system many algorithms with different characteristics and computational requirements are used in a sequence where output of one algorithm becomes the input of the next algorithm in the sequence [1,2]. An example of such a system is a motion estimation systems. In such a system, a sequence of images of a scene are used to compute the motion parameters of a moving object in the scene. Figure 1 shows the computational flow for a motion estimation system in which stereo images (L_{im} and R_{im}) at each time frame are used as the input to the

This research was supported in part by National Aeronautics and Space Administration Under Contract NASA NAG-1-613, and in part by National Science Foundation Grant IRI 87-05400.

system. Briefly, the involved tasks (or algorithms) in this system are as follows. The first algorithm is computation of zero crossings of the images (edge detection (L_{zc} and R_{zc})). The zero crossings are used as feature points for both stereo and time matching. The stereo match algorithm provides points to compute 3-D information about the object in the scene. Using these matched points (Lsm and Rsm), the corresponding points in the image in the next time frame (L_{tm}) are located and this task is performed by time match algorithm. Again, stereo match is used to obtain the corresponding 3-D points in the next image frame. These two sets of points provide information to compute the motion parameters. The above process is repeated for each new set of input image frame.

The computational requirements for such vision systems are tremendous [2, 3]. Not only does such a system requires powerful parallel processing capabilities but, to obtain any significant speedups and performance gains from parallel processing over sequential processing, efficient data decomposition and load balancing need to be employed at each step in the system.

First, this paper presents algorithms for a motion estimation system. The algorithms are non-iterative, and obtain matched features points among stereo images at any two consecutive time instants. The system consists of the following steps: 1) extraction of features, 2) stereo match of images in one time instant, 3) time match of images from consecutive time instants, 4) stereo match to compute final unambiguous points

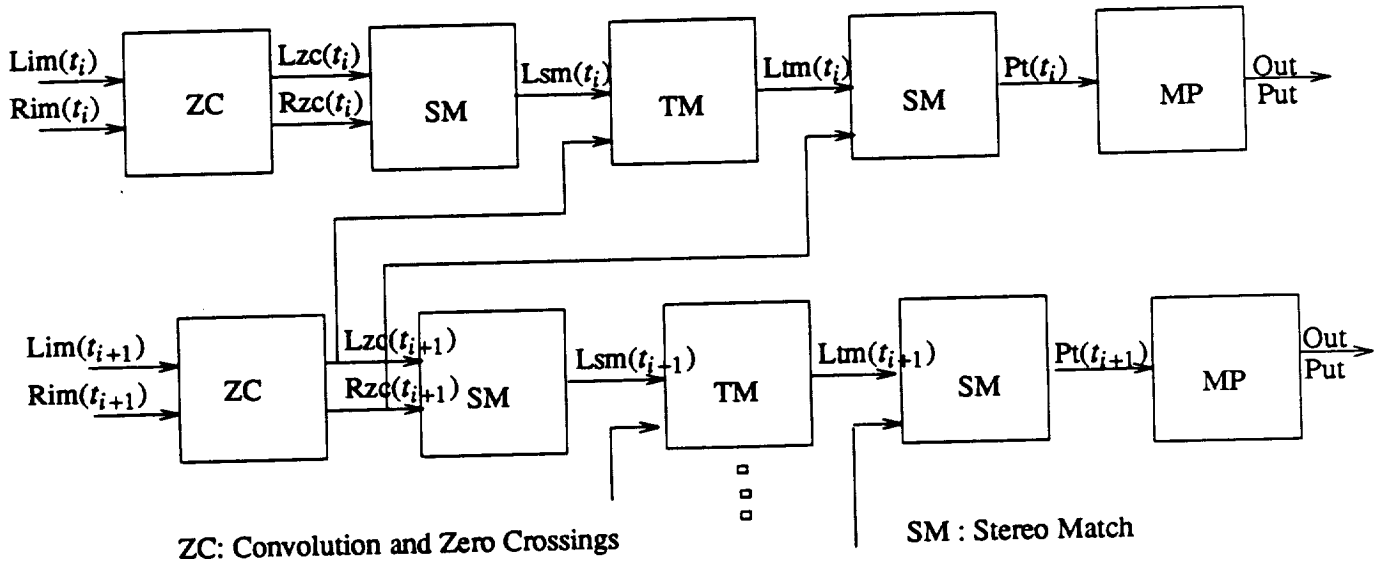


Figure 1 : Computation flow for motion estimation

and, 5) computation of motion parameters. Since zero crossings points are used as the features for matching, there is less data involved in the matching process so that considerable amount of saving in the computation times can be achieved in solving the correspondence problem. The algorithms are applied to real outdoor images, and are shown to perform well.

Second, we present techniques to perform efficient data decomposition and load balancing for vision systems for medium to large grain parallelism. The important characteristics of these techniques are that they are general enough to apply to most vision systems, and they use statistics and knowledge from execution of the preceding task to perform data decomposition and load balancing for the current task. For example, in the motion estimation system sufficient knowledge can be obtained about the output data from the zero crossing step to perform data decomposition and load balancing for the stereo matching step. The advantages of such techniques are: First, they use characteristics of the involved tasks and data, and therefore, work well no matter how data changes. Second, many vision systems consist of such tasks, and exhibit the above described computation flow, and therefore, these techniques can be used in many systems (e.g., object recognition, optical flow etc.) [2].

Finally, the performance of the proposed techniques is evaluated by using a parallel implementation of the motion estimation system algorithms on a hypercube multiprocessor system. The results show that using uniform partitioning without considering the computations involved, parallel processing does not provide significant performance improvements over sequential processing. Furthermore, by applying the proposed data decomposition and load balancing techniques, significant performance gains (as much as 6 fold) can be obtained over uniform partitioning.

This paper is organized as follows. In Section 2 we present the algorithms for each step in the motion estimation system [4]. These algorithms will provide insight into the involved computations in the system and provide a framework for the discussion in the following sections. Section 3 contains the proposed load balancing and data decomposition techniques. Some examples are also presented to illustrate the techniques. In Section 4 we present parallel implementation of these algorithms on a hypercube (Intel iPSC/2) multiprocessor. We discuss the performance results for each of these algorithms as well as present the performance of the data decomposition, and load balancing schemes. Some of these techniques have been applied to other vision systems and have been shown to work well [5]. Finally, concluding remarks are presented.

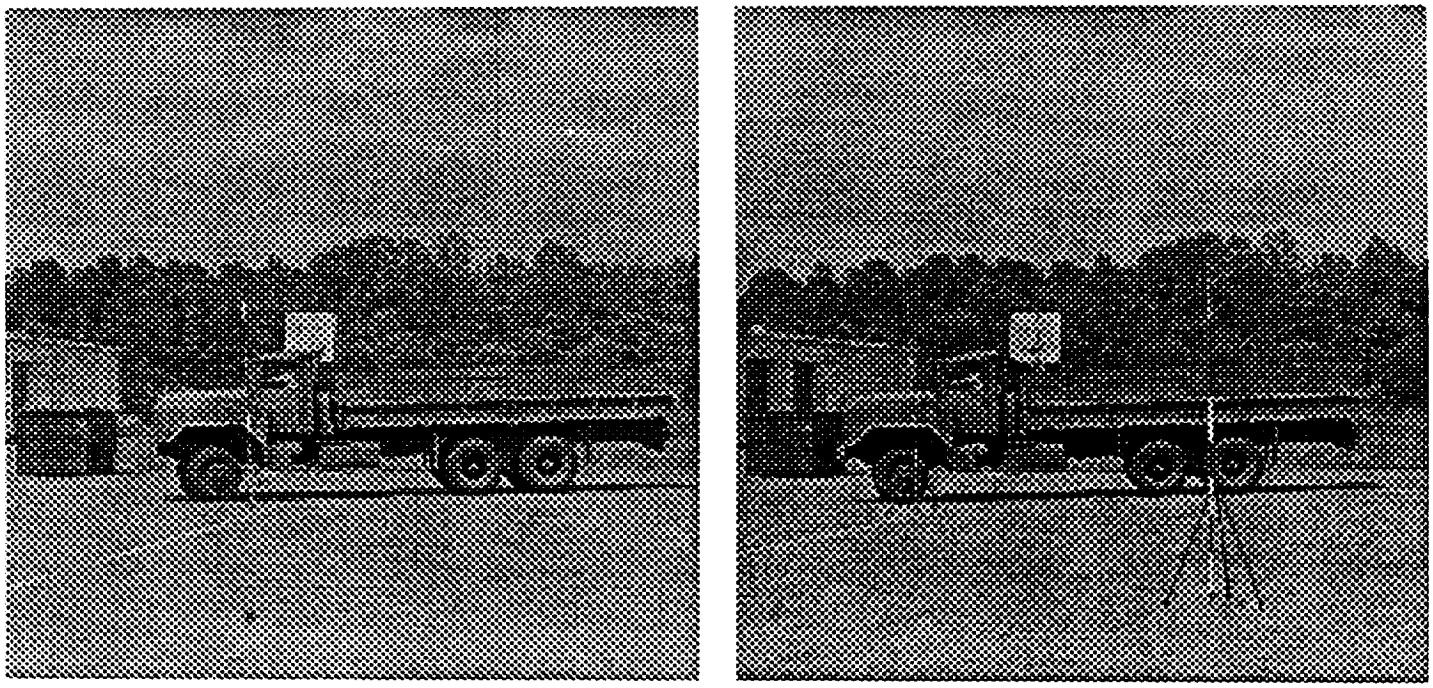
2. Motion Estimation Algorithms

This Section describes the steps in the motion estimation system. A detailed description of the involved computations is included in order to understand the characteristics of such algorithms. In general, the problem of motion estimation involves two sub-problems which are 1) matching feature points between images and 2) solving the motion parameters based on the point correspondences. In this paper, we will not discuss the last process, calculation of motion parameters, and a discussion on techniques to compute motion parameters can be found in [6]. Nevertheless, we simply use the techniques to solve the motion parameters in the last process of our algorithms.

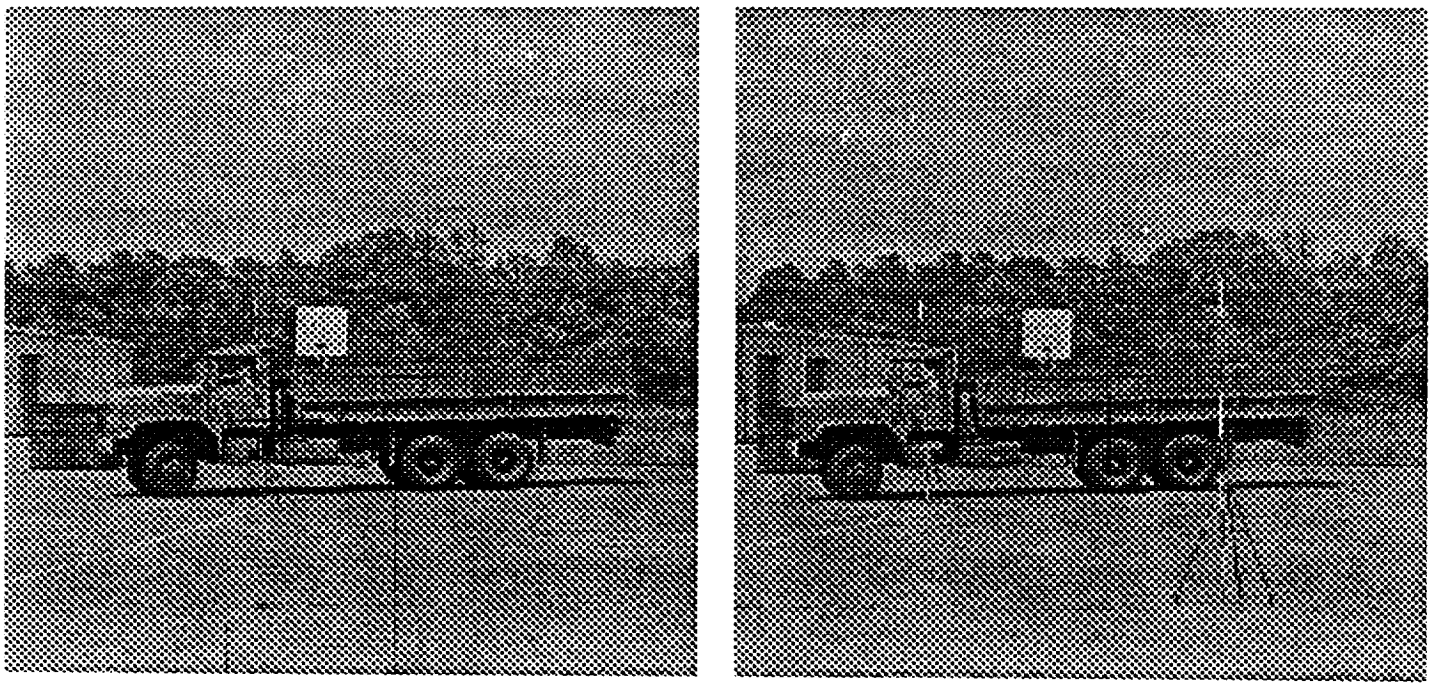
The matching algorithm is used to find point correspondences in pairs of stereo images (of size 256×256) at two consecutive time instants (t_{i-1} and t_i). Typical stereo image pairs at two consecutive time instants (t_7 and t_8) used in this paper are shown in Figure 2, which are outdoor scenes of a truck at different locations. These images are segmented out from the corresponding larger images of size 1024×1024. The imaging setup employed in taking the images is the parallel axis method as shown in Figure 3. The algorithm consists of two major processes which are 1) extracting feature points and 2) matching. The feature points used in the matching process are edge points which are considered as the more reliable features obtained from an image. The matching process is done by employing non-iterative procedures with the heuristic of limited displacement (or disparity) between frames. The use of edge points and non-iterative procedures with the limited displacement (or disparity) assumption saves a considerable amount of computation in solving the correspondence problem.

2.1. Feature Points

The feature points used in this algorithm are zero crossing points of an image. We employ the method suggested by Huertas and Medioni [7] to extract the zero crossings of an image. In their method, they decomposed the 2-D Laplacian-of-Gaussian mask into a sum of two separable filters which was used to convolve with an image. Then, they used 11 predicates, which defined a total of 24 edge positions, to determine edge locations. In order to eliminate non-significant zero crossing points and maintain enough details, we threshold the zero crossing image based on the intensity gradient at each zero crossing point. Figure 4 depicts the thresholded zero crossing images of the pictures shown in Figure 2.



(a) Left and right images at time instant t_7



(b) Left and right images at time instant t_8
Figure 2 : Images set of t_7 and t_8

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

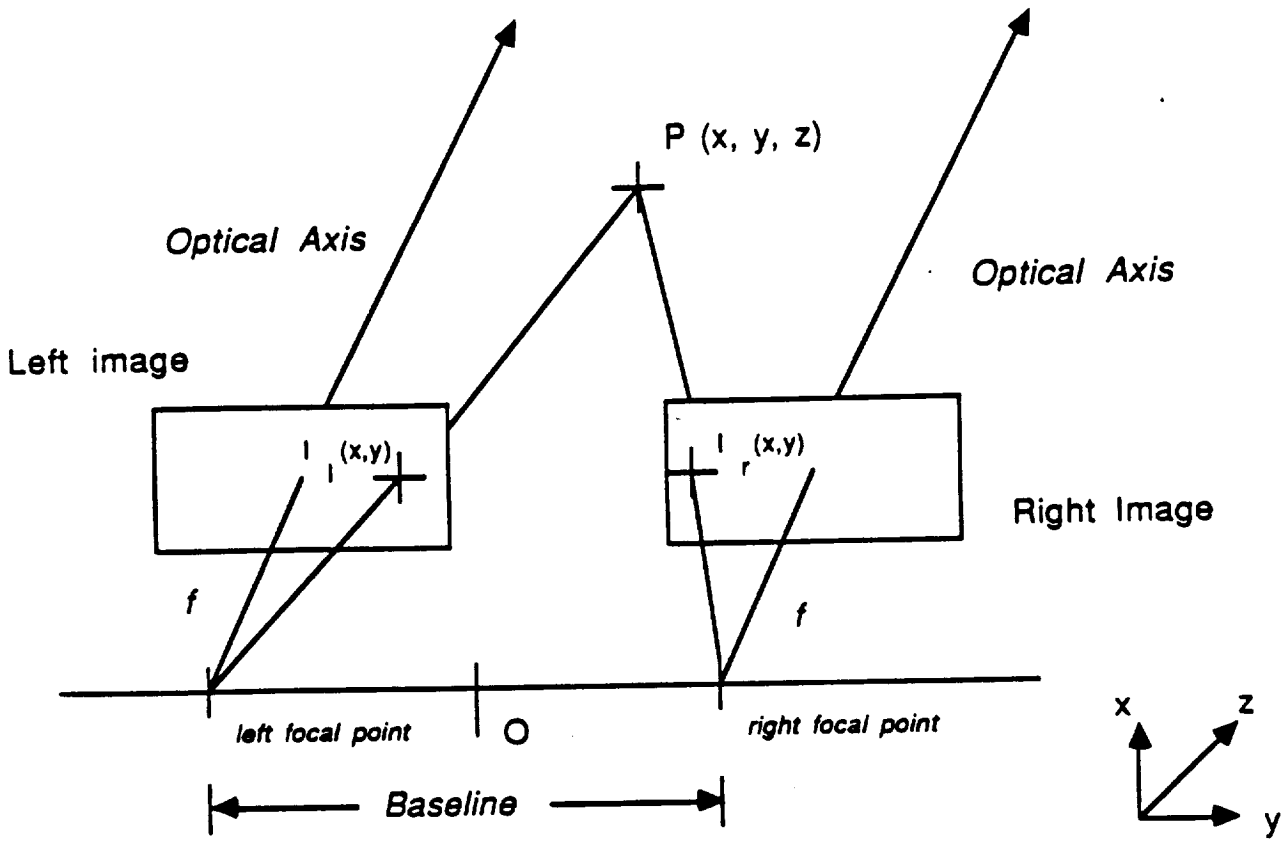
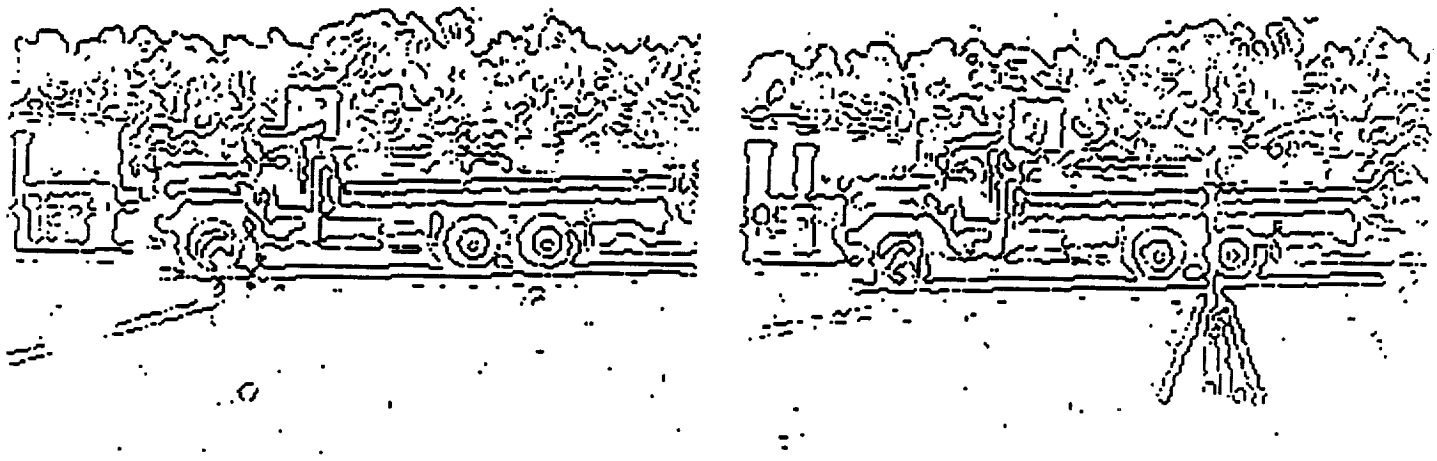
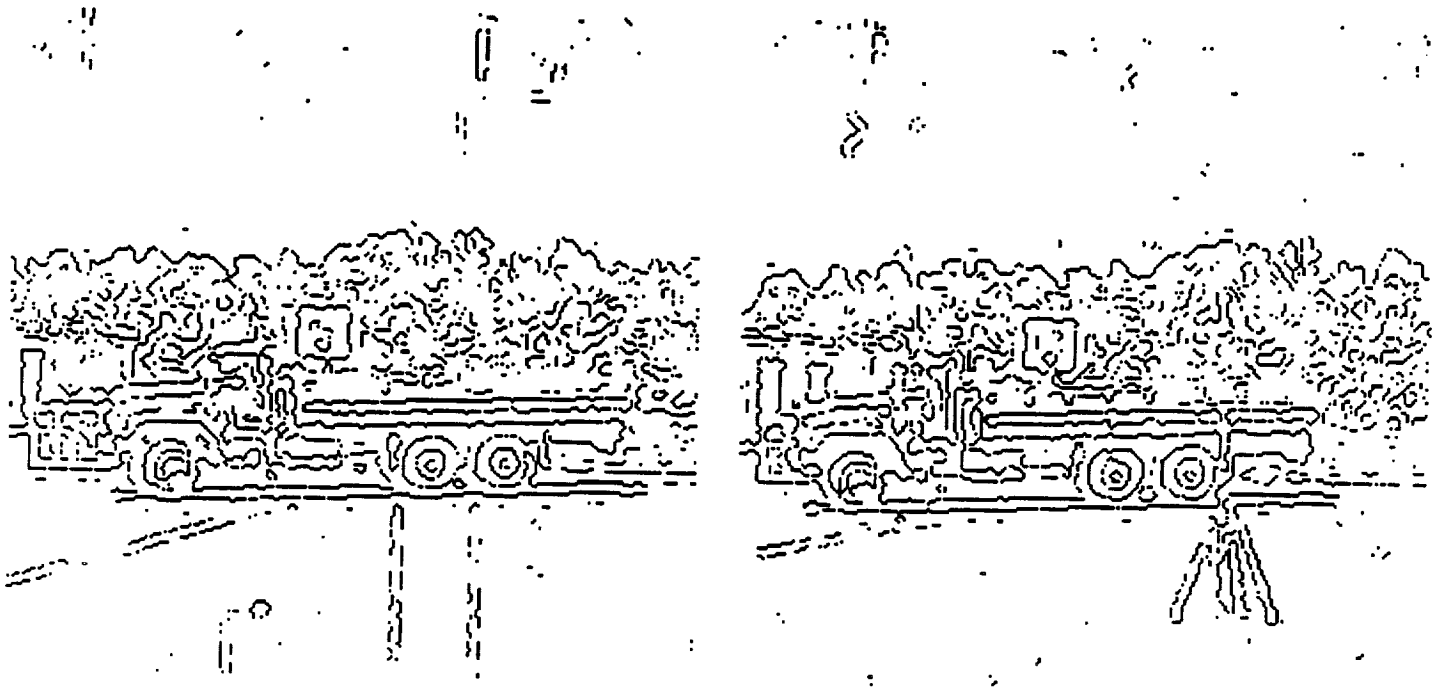


Figure 3 : Imaging geometry of the parallel axis method

ORIGINAL PAGE IS
OF POOR QUALITY



(a) Left and right zero crossings at time instant t_7



(b) Left and right zero crossings at time instant t_8

Figure 4 : Zero crossings of images in Figure 2, w = 5 pixels and threshold =70

Having obtained the zero crossings of an image, we associate each zero crossing point with one of the sixteen possible zero crossing patterns as shown in Figure 5. The use of these sixteen zero crossing patterns was first suggested and used by Kim and Aggarwal in their point correspondence algorithm [8]. Similar to their algorithm, we use all these possible zero crossing patterns, except pattern (b) in Figure 5, as the matching features. The horizontal pattern (pattern (b)) is excluded in the stereo matching because the search for matching is done on the same scan line and causes ambiguous matches. Consequently, in order to have consistent matching features, the zero crossing points with the same pattern (pattern (b)) are also excluded in the time matching process. The patterns are not used directly; instead, we assign each pattern a value (as suggested in [8]) according to its local connectivity. These pattern values are useful to measure the similarity of zero crossing patterns and are used in the matching process. The values are calculated as follows:

a) For a zero crossing point location (p,q), its eight neighbors are numbered as shown in Figure 6.

b) The value of a given zero crossing point (forming one of the sixteen patterns) is equal to the sum of the two numbers corresponding to its two attached neighbors.

Examples :

The value of Pattern (d) is $1 + 3 = 4$.

The value of Pattern (j) is $2 + 6 = 8$.

e) If a zero crossing point has only one attached neighbor or none at all, the assigned value is 20.

The similarity between any two zero crossing points is based on the directional difference of their zero crossing patterns. For example, the directional difference between patterns (a) and (e) in Figure 5 is 2, and the difference between patterns (a) and (g) is 4. The directional difference between any two direction values (e.g. D_1 and D_2) is calculated as follows:

$$DIFF = | D_1 - D_2 |$$

$$\text{if } (DIFF > 4), DIFF = | 8 - DIFF |$$

In our matching process, the use of directional difference (or zero crossing pattern values) in finding matched point pairs is through the expression of directional difference weight as shown below:

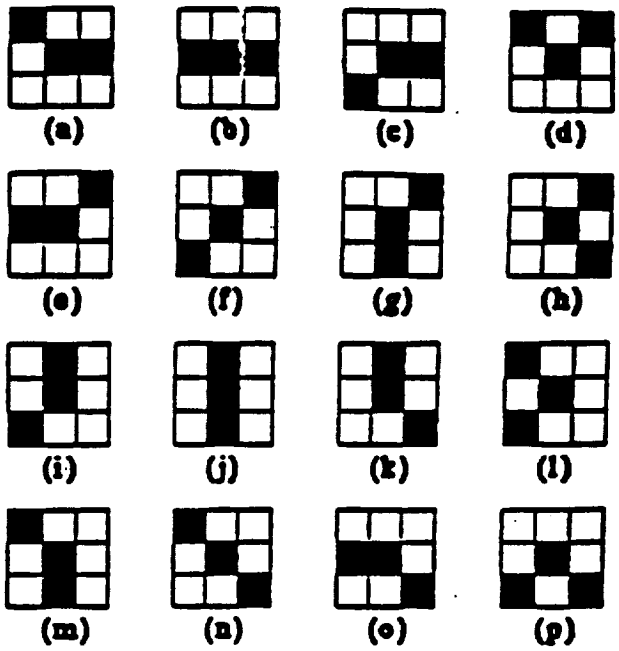


Figure 5 : Zero crossing patterns

3	2	1
4	(p,q)	0
5	6	7

Figure 6 : Value assignment for a zero crossing pattern

$$W_{ddif} = \frac{1}{1 + DIFF} \tag{1}$$

2.2. Matching

Once zero crossings are extracted in all the involved images, the matching process is applied to find point correspondences among the images (two stereo image pairs at two consecutive time instants, i. e. t_{i-1} and t_i). The evidences used in this process to obtain matched point pairs are the normalized correlation coefficient and the directional difference weight [8]; furthermore, in order to limit the search space, the heuristic of limited displacement or disparity between frames is exploited. The matching process consists of six steps as follows:

1) Perform stereo (from left to right) matching in the t_{i-1} stereo image pair.

2) Obtain unambiguous matched point pairs by eliminating multiple matches.

3) Perform time matching between the unambiguous matched points in the left t_{i-1} image and the feature points of the left t_i image.

4) Obtain unambiguous matched point pairs from the time matched points by eliminating multiple time matches.

5) Perform stereo matching between the unambiguous matched points (obtained in step (4)) in the left t_i image and the feature points of the right t_i image.

6) Obtain unambiguous matched point pairs from the results of t_i stereo matching by eliminating multiple matches.

The results of the above steps are two sets of unambiguous stereo matched point pairs at time instant t_{i-1} and t_i. These two sets are related through steps (3) and (4), the matching over time; therefore, we can pick out all the unambiguous matched points that correspond to each other among the two stereo image pairs at time instants t_{i-1} and t_i. In the remaining sub-section, we are going to discuss the processes in detail. The discussion is divided into three major sub-sections which are: i) stereo matching, ii) time matching and iii) elimination of multiple matches.

2.2.1. Stereo Matching

This is the sub-process to obtain the matched point in the right image for each matchable zero crossing point in the corresponding left image of the same stereo pair. Since the imaging setup is the parallel axis method, we exploit the epipolar line constraint in solving the stereo matching problem. As the result, we have 1-dimensional search space instead of 2-dimensional search space in the stereo matching process. Figure 7 shows a typical search space in the right image for a matchable zero crossing point in the left image; the search space is on the left side of the transferred location of that particular left image zero crossing point. The d_{max} in Figure 7 is the maximum possible disparity (heuristic of limited disparity between frames) between the left and right images.

Let S_{pt} be the set of all non-horizontal zero crossing points in the right image within the search space of a zero crossing point in the left image. The stereo matching process is as follows:

For each point in S_{pt},

i) Calculate the normalized correlation coefficient with a template size of $a \times a$ between the grey level images of left and right at the corresponding locations. The normalized correlation coefficient is calculated by using the following expression :

$$\rho_s = \frac{\sum_i \sum_j (lx_{ij} - \overline{lx})\,(rx_{ij} - \overline{rx})}{\sqrt{\sum_i \sum_j (lx_{ij} - \overline{lx})^2}\ \sqrt{\sum_i \sum_j (rx_{ij} - \overline{rx})^2}} \qquad (2)$$

where

lx_{ij} is the grey value at point (i,j) in the left image.

rx_{ij} is the grey value at point (i,j) in the right image.

$\overline{lx}$ is the mean grey value of the template in the left image.

$\overline{rx}$ is the mean grey value of the template in the right image.

ii) If the normalized correlation value ρ_s is less a threshold value $thrsh_{\rho_s}$, we discarded that particular point in the search space in the remaining steps.

iii) Calculate the directional difference weight ($w_{ddif\,(stereo)}$) between the left and the right zero crossing point (within the search space) according to Equation (1).

iv) Obtain the total weight as the combination of the correlation coefficient and the directional difference weight.

$$w_s = a \times \rho_s + b \times w_{ddif\,(stereo)}, \qquad where\ a + b = 1. \tag{3}$$

v) Among all elements of S_{rl}, the point with the maximum total weight w_s is considered as the matched point for the corresponding zero crossing point in the left image.

2.2.2. Time Matching

This is the sub-process to obtain the matched point in the left t_i image for each candidate zero crossing point in the corresponding left t_{i-1} image. Similar to the stereo matching process, we exploit the heuristic of limited displacement (instead of disparity) between frames in solving the time matching problem. We assume that the total motion between the t_{i-1} and t_i frames is within f pixels in vertical direction and h pixels (from right to left) in horizontal direction. Hence, the search space for each candidate zero crossing point in the left t_{i-1} image is a window of size $f \times h$ pixels on the left size of the transferred location in the left t_i image as shown in Figure 8. Any zero crossing point (except horizontal ones) inside this window is a potential match point for the corresponding candidate zero crossing point in the left t_{i-1} image. The time matching process is as follows:

For each non-horizontal zero crossing point in the left t_i image within the search space of a zero crossing point in the left t_{i-1} image,

i) Calculate the normalized correlation coefficient with a template size of $t \times t$ between the grey level image of t_{i-1} and t_i at the corresponding locations. The normalized correlation coefficient is calculated by using the following expression :

$$\rho_t = \frac{\sum_i \sum_j (x_{ij} - \bar{x})\,(y_{ij} - \bar{y})}{\sqrt{\sum_i \sum_j (x_{ij} - \bar{x})^2}\ \sqrt{\sum_i \sum_j (y_{ij} - \bar{y})^2}} \tag{4}$$

where

x_{ij} is the grey value at point (i,j) in the t_{i-1} image.

y_{ij} is the grey value at point (i,j) in the t_i image.

$\bar{x}$ is the mean grey value of the template in the t_{i-1} image.

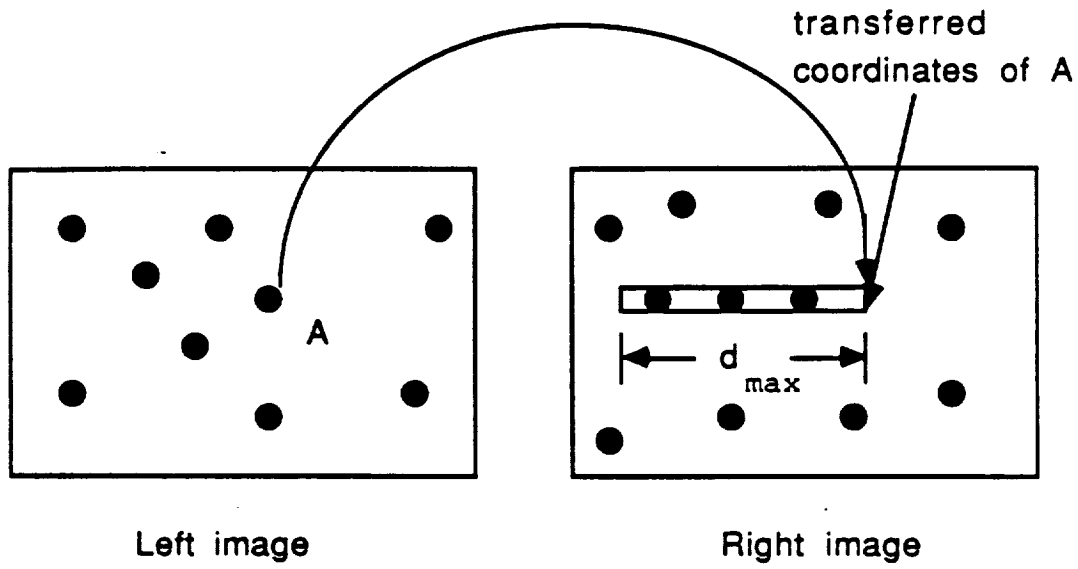


Figure 7 : Search space for stereo matching

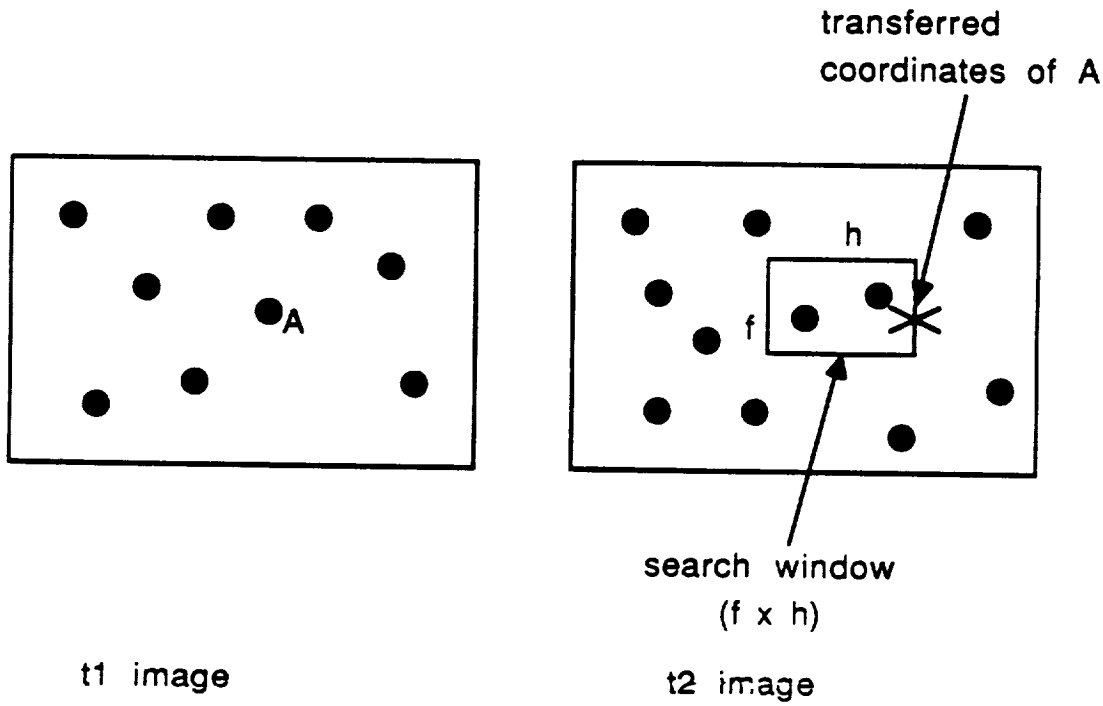


Figure 8 : Search space for time matching

$\bar{y}$ is the mean grey value of the template in the t_i image.

ii) If the normalized correlation coefficient ρ_t is less than a threshold value $thrshd_{\rho_t}$, we discard that particular point in the remaining steps.

iii) Calculate the directional difference weight ($w_{ddif\,(time)}$) between the left t_{i-1} and the left t_i zero crossing point (within the search space) according to Equation (1).

iv) Obtain the total weight as the combination of the correlation coefficient and the directional difference weight.

$$w_t = c \times \rho_t + d \times w_{ddif\,(time)}, \qquad where\ c + d = 1. \tag{5}$$

v) Within a search window in the left t_i image, the zero crossing point with the maximum total weight w_t value is considered as the match point for the corresponding zero crossing point in the left t_{i-1} image.

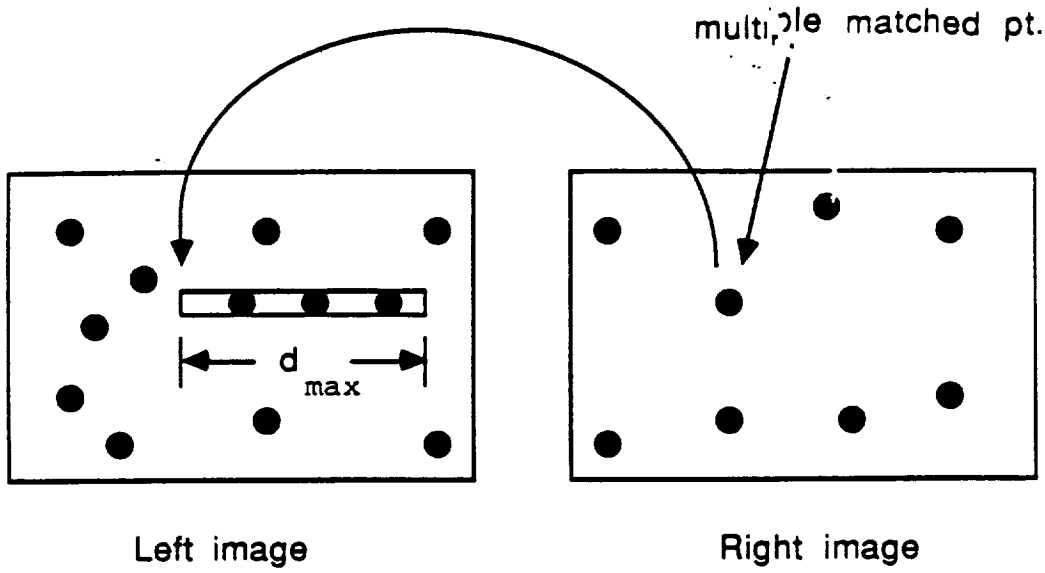
2.2.3. Elimination of Multiple Matches

After the sub-process of either stereo matching or time matching, there may be multiple matches for some zero crossing points in either the left image or the left t_{i-1} image. In this paper, we use the same procedure in eliminating both types (stereo or time) of multiple matches except different sizes of the search window are exploited. For stereo matching, we use a 1-D search window of size $d_{\max}$ as shown in Figure 9(a); on the other hand, for time matching, we use a search window of size $f \times h$ as shown in Figure 9(b). The remaining steps of the procedure for determining unambiguous matched points are as follows:

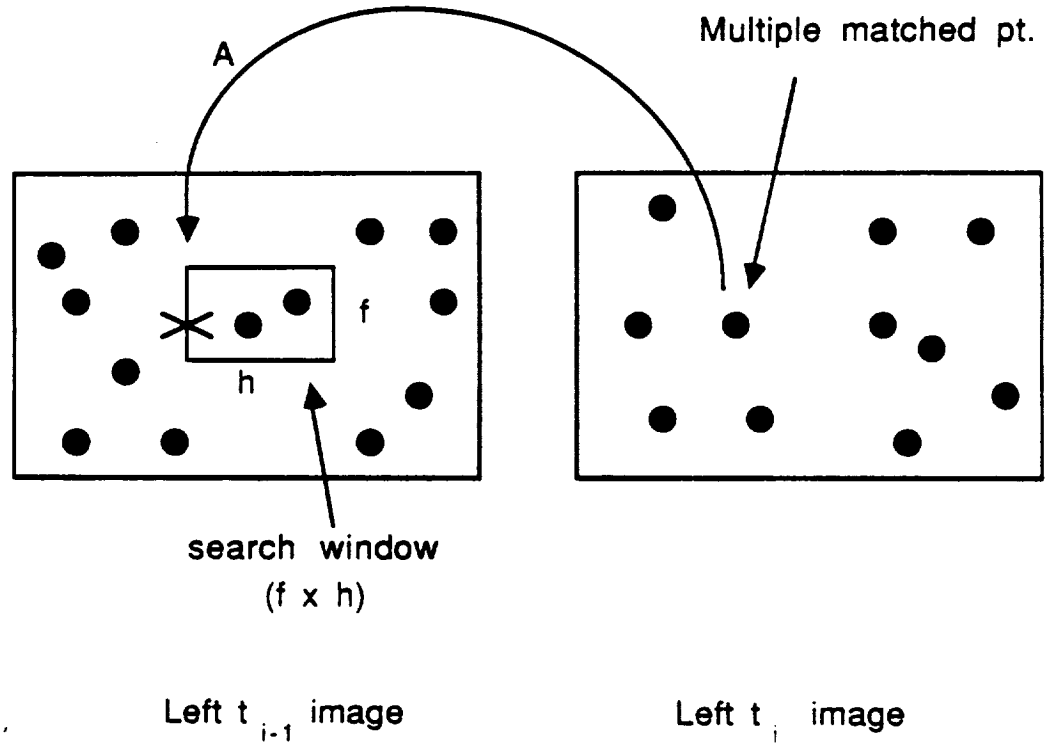
i) At the position of a multiple match in either the left image for stereo matching or the left t_{i-1} image for time matching, open a search window either with a size of $d_{\max}$ or with a size of $f \times h$ respectively.

ii) Within the search window, locate all the positions that have the same (multiple) match.

iii) If all the positions are within the neighborhood region r_{neigh}, calculate the total weight (w_s or w_t) according to Equation (3) or (5) (depending on whether we try to eliminate stereo multiple matches or time multiple matches). The position with the highest value in its total weight is regarded as the correct match.



(a) Stereo case



(b) Time case

Figure 9 : Search windows for elimination of multiple matches

iv) If one or more positions are outside the neighborhood region r_{neigh}, we use the disambiguation procedure described in Section 2.2.3.1 to resolve the multiple matches.

v) If multiple matches still exist after the application of the above steps, they all are discarded from the match set.

2.2.3.1. Disambiguation

We only use this procedure, if step (iv) in the above discussion is true. In this procedure, the neighboring unambiguous matched points around a multiple matched point are used as one of the supporting evidences in determining the correct match. The other evidences used are the normalized correlation coefficient and the directional difference weight. The steps are as follows:

i) At each position, calculate the normalized correlation coefficient (ρ_s for stereo matching or ρ_t for time matching) and the directional difference weight ($w_{ddif\,(stereo)}$ for stereo matching, or $w_{ddif\,(time)}$ for time matching).

ii) Assign a correlation coefficient rank, R_{cc}, and zero crossing pattern rank, R_{zcp}, to each position according to its normalized correlation coefficient (ρ_s or ρ_t) and directional difference weight ($w_{ddif\,(stereo)}$ or $w_{ddif\,(time)}$). The position with the highest value in normalized correlation coefficient or directional difference weight has the highest rank in R_{cc} or R_{zcp}.

iii) At each position, check for unambiguous matched neighbors. If it has two attached unambiguous matched neighbors, a neighbor weight $neigh_{wt}$ of 3 is assigned. On the other hand, if it has only one attached unambiguous matched neighbor, a neighbor weight $neigh_{wt}$ of 2 is assigned.

iv) At each position, open a check window of size 5×5 but excluding the center 3×3 region as shown in Figure 10 and count the number of unambiguous matched points, num_{smp}.

v) At each position, calculate the total possibility as the sum of the ranks, the weight and the number.

$$poss_{tot} = R_{cc} + R_{zpw} + neigh_{wt} + num_{smp} \tag{6}$$

vi) The position with the highest $poss_{tot}$ value is considered as the correct match.

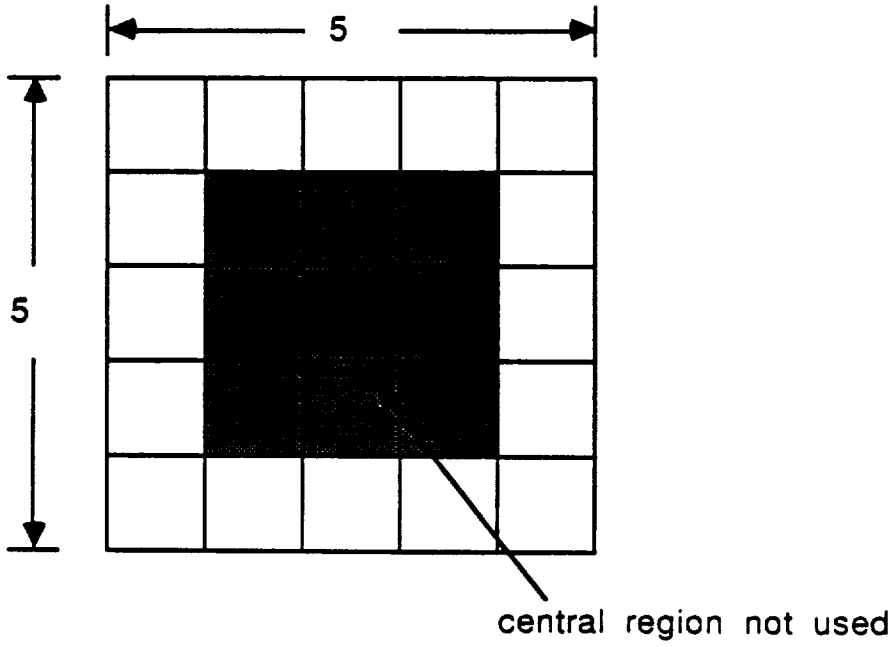


Figure 10 : Check window for disambiguation

2.3. Experimental Results

This algorithm has been tested on two stereo image pairs shown in Figure 2. The zero crossings are first extracted from the images as the feature points and the results are shown in Figure 3. We assign a zero crossing pattern value to each zero crossing point; then, the matching procedures are applied to the image pairs. In our experiment, we have the following assumptions on the images :

a) The maximum possible disparity d_{max} is assumed to be 20 pixels.

b) The total motion between time frames is $f = 5$ pixels in vertical motion and $h = 20$ pixels in horizontal motion.

c) The template size used in the stereo matching is 5×5 pixels ($a = 5$ pixels).

d) The template size used in the time matching is 10×10 pixels ($t = 10$ pixels).

e) Both $thrshd_{\rho_s}$ and $thrshd_{\rho_t}$ are equal to 0.55.

f) We treat both the normalized correlation coefficient (ρ_s or ρ_t) and the zero crossing pattern weight (zcw_s or zcw_t) equally; therefore, $a = b = c = d = 0.5$ are used in both Equations (3) and (5).

g) In the procedure of elimination of multiple matches, the neighborhood region r_{neigh} is 5 pixels for stereo matching and 5×5 pixels for time matching.

Figure 11 shows the stereo matching results of the t_{i-1} (t_7) stereo image pair. We can observe that there are more matched points in the left image than in the right image. The extra points in the left image are due to multiple matches. In order to eliminate the the multiple matches, we apply the procedures of elimination of multiple matches to the t_7 stereo matched points and the results are shown in Figure 12. Then, using these unambiguous matched points in the left t_{i-1} (t_7) image as the candidate points, we match them with the feature points of left t_i (t_8) image by using the time matching procedures and the results are depicted in Figure 13. The unambiguous time matched points after the elimination of multiple matches are shown in Figure 14. Having the unambiguous matched points in the left t_8 image, we match them with the feature points of right t_8 image and Figure 15 shows the unambiguous matched points after the elimination of multiple matches. With these two sets of unambiguous stereo matched points at t_7 and t_8, we can pick out all the unambiguous matched points that correspond to each other among the four images. The results are depicted in Figure 16. From these two sets of stereo matched points, we can calculate the 3-D position of each matched point at t_7 and t_8 by using triangulation. The motion parameters can be estimated based on the 3-D points from t_7 and t_8.

3. Data Decomposition and Load Balancing Techniques

In a multiprocessor system the simplest method to implement a task in parallel is to decompose the data and equally and uniformly among the processors. In a completely deterministic computation in which the computation is independent of the input data such schemes perform well, and normally, the processing time is comparable on all the processors. That is, efficient utilization and load balancing can be obtained. For example, regular algorithms such as convolutions, filtering or FFT exhibit such properties. The amount of computation to obtain each output point is the same across all input data. Therefore, uniform decomposition of data results in load balanced implementation.

Most other algorithms do not exhibit a regular structure, and the involved computation is normally data dependent. Furthermore, the computation is not uniformly distributed across the input domain. In such cases, a simple decomposition of data does not provide efficient mapping, and results in poor utilization and low speedups. Also, the performance cannot be predicted for a given number of processors, and a

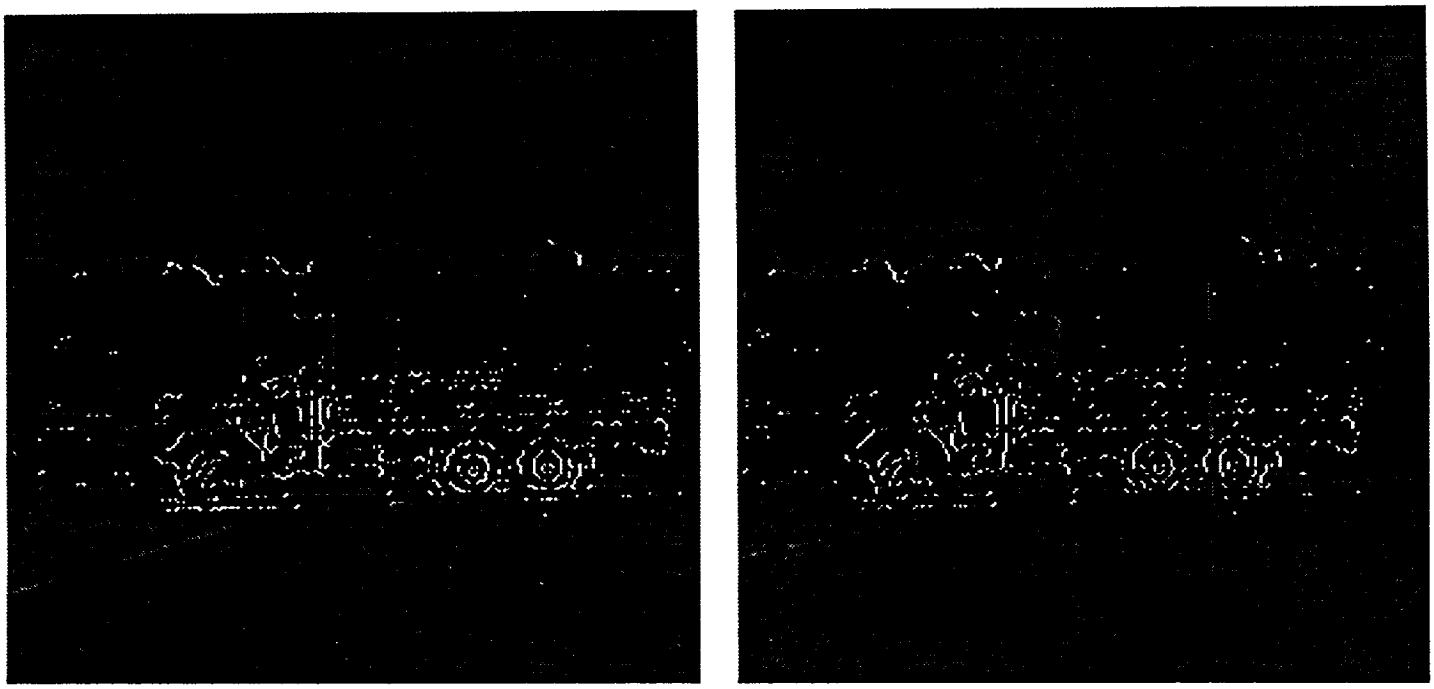


Figure 11 : Results of stereo matching of t_7 stereo image pair

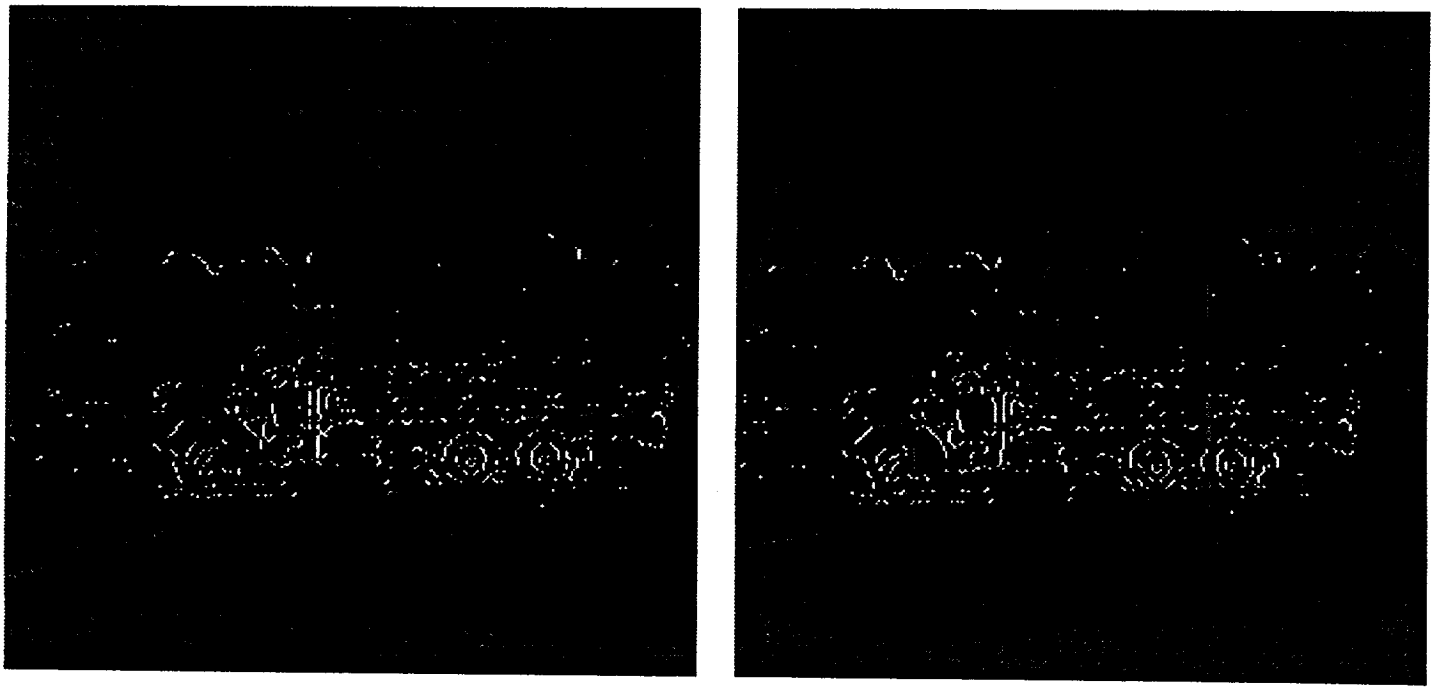


Figure 12 : Unambiguous matched points of the t_7 stereo image pair

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

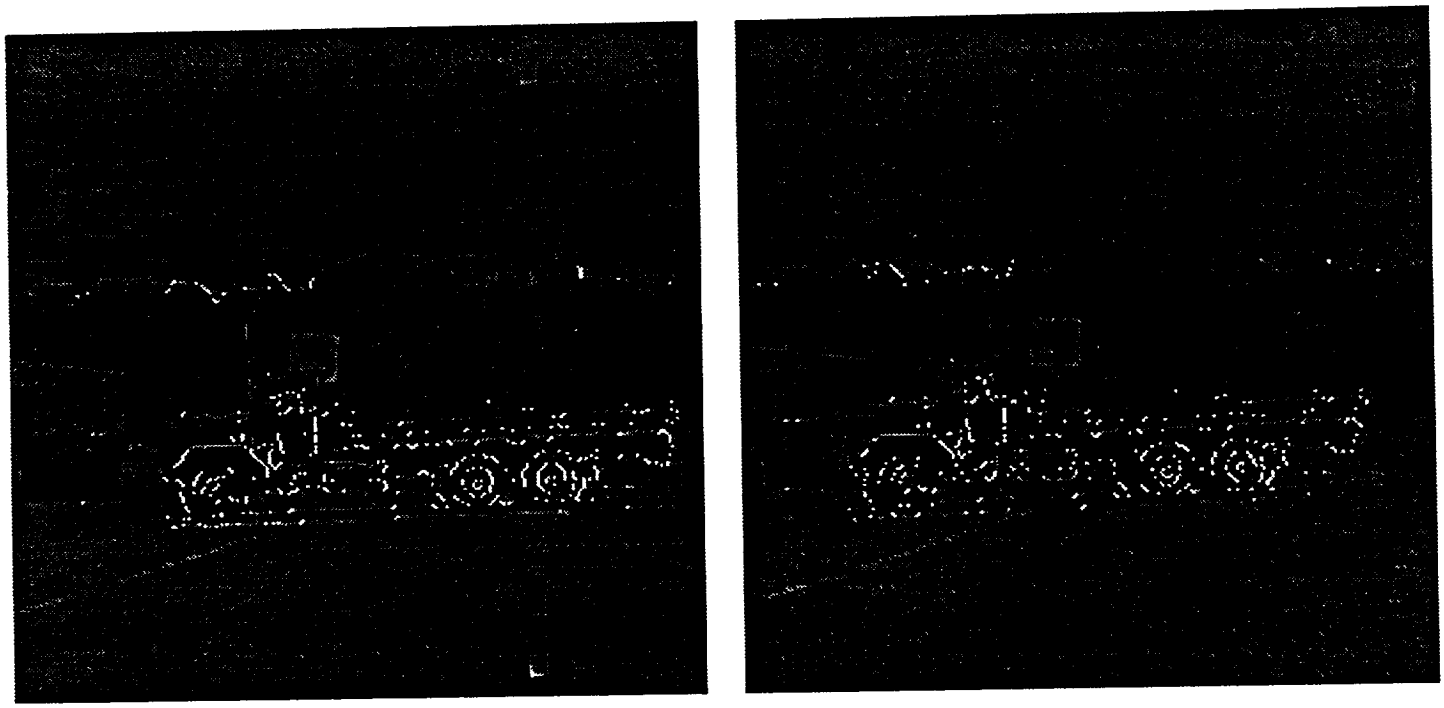


Figure 13 : Results of time matching between l_7 and l_8

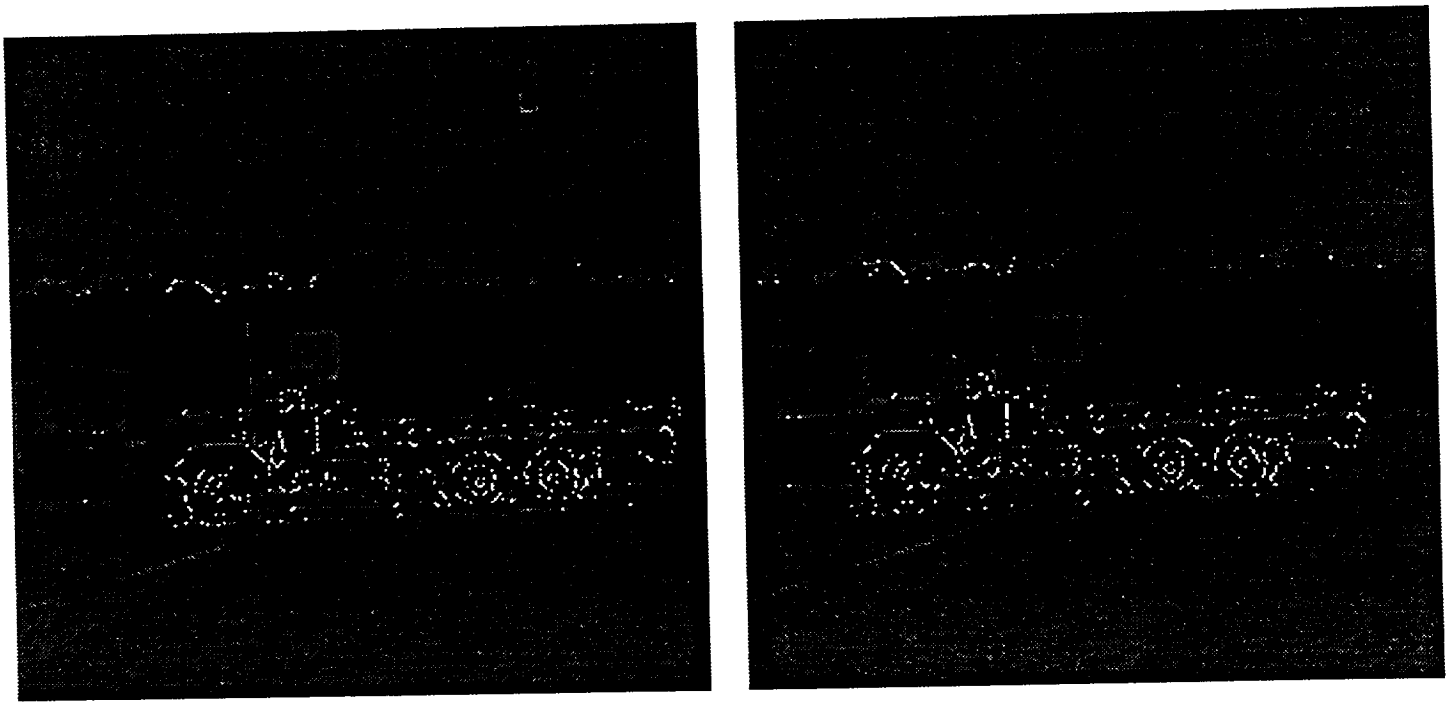


Figure 14 : Unambiguous time matched points of the t_7

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

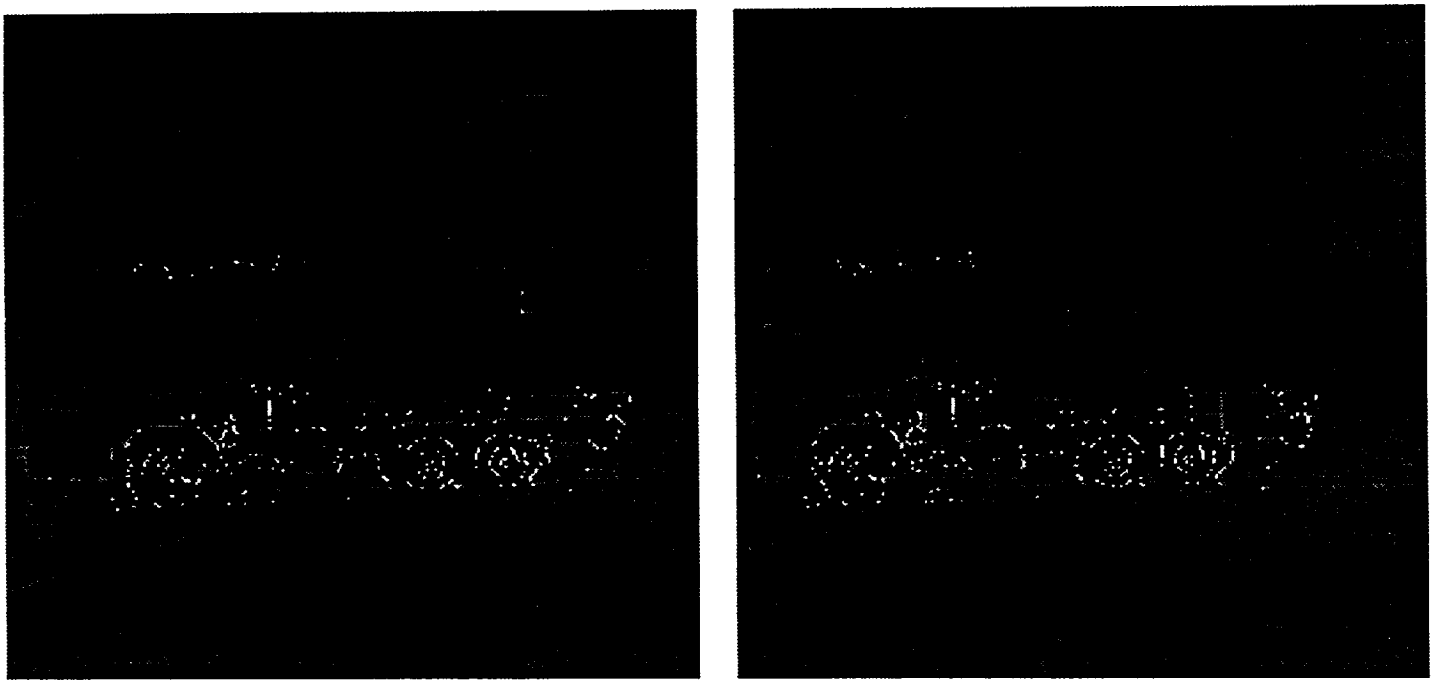
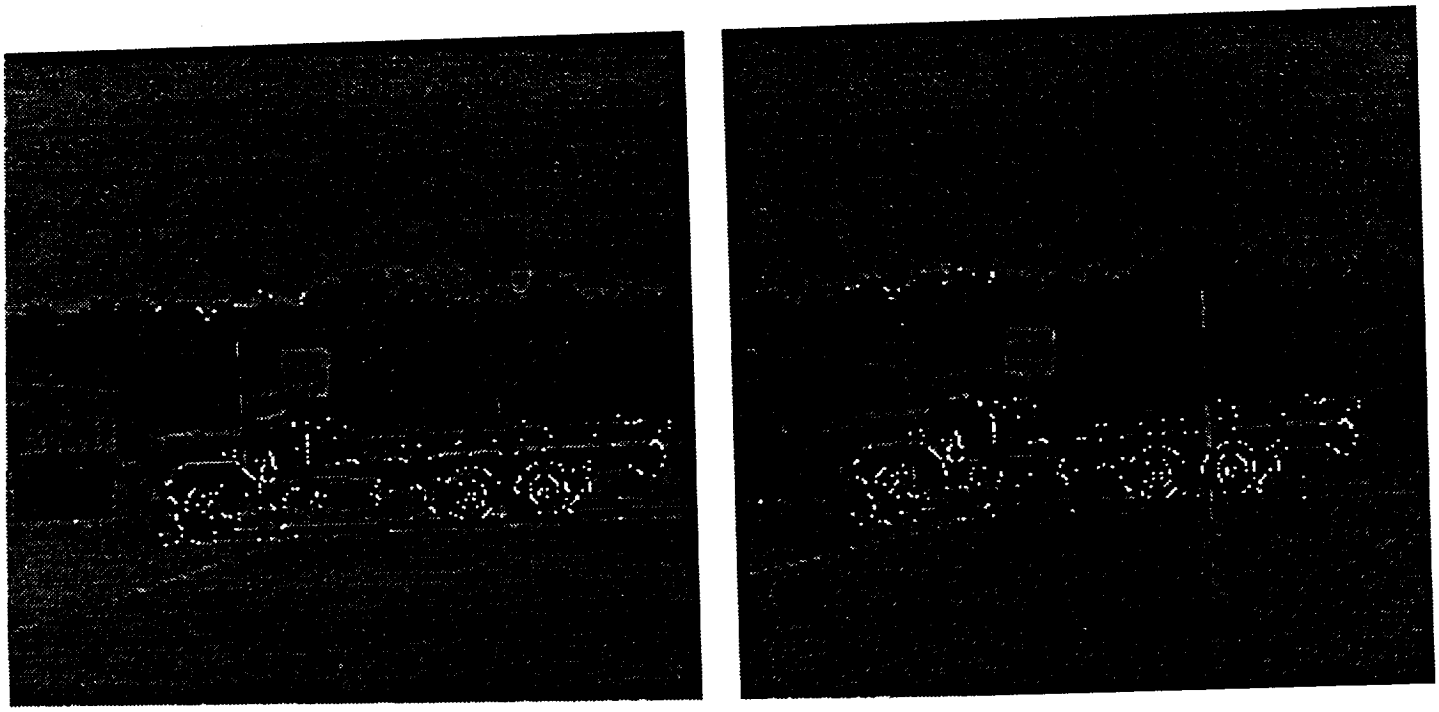
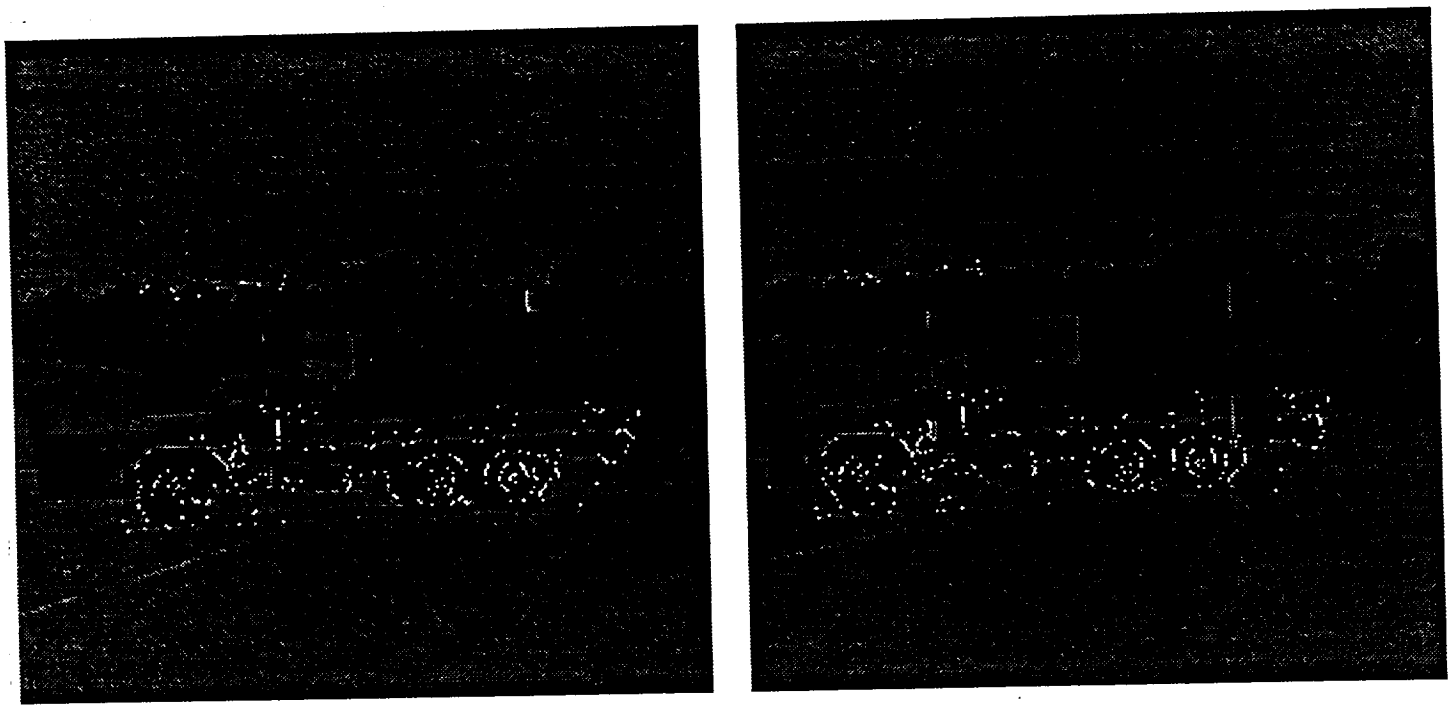


Figure 15 : Unambiguous matched points of matched points of the t_8 stereo image pair (after time matching with l_7)

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH



(a) : At time instant t_7



(a) : At time instant t_8

Figure 16 : Unambiguous matched points of Figure 2

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

given data size, because the computation varies as type of data and its distribution varies. For example, in the stereo match algorithm, the computation is more where feature points are dense, and is comparatively small where number of features is small and sparsely distributed (Figure 4).

In the algorithms presented in the previous section, the data structures and and computations can be divided into two categories; namely, regular and uniform; and irregular and non-uniform. In the first category, computation is uniformly distributed across the input domain, and it is data independent. For example, feature extraction involves regular and uniform computations. On the other hand, all the matching algorithms are data dependent and the computation has a non-uniform distribution across the input domain. For example, in stereo and time matching algorithms, computation depends on the number of features (which is image data dependent), and distribution of features (spatial relationship of the features). The computation is more where features are densley distributed compared to where same number of features are sparsley distributed.

In a vision system, it is important to efficiently allocate resources and perform load balancing at each step to obtain any significant performance gains overall. An important characteristic of such systems is that the input data of a task is the output of the previous task. Therefore, while computing the output in the previous task enough knowledge about the data can be obtained to perform efficient scheduling and load balancing.

Consider a parallel implementation of a task on n processor parallel machine. Let T_i $(1 \le i \le n)$ denote the computation time at processor node i. Then the overall computation time for the task is given by

$$T_{\max} = \max\{T_1, \ldots, T_n\} \tag{7}$$

The total wasted time (or idle time) T_w is given by

$$T_w = \sum_{i=1}^{i=n} (T_{\max} - T_i) \tag{8}$$

If $T_{\max} = T_i$ for all i, $1 \le i \le n$, then the task will be completely load balanced. Another measure of imbalance is given by the variation ratio V,

$$V = \frac{T_{\max}}{T_{\min}}, \quad T_{\min} = \min\{T_1, \ldots, T_n\} \tag{9}$$

The goal in performing load balancing is to minimize T_w, or move V as close to 1 as possible. In the best case, $T_w = 0$ or $V = 1$.

If T_{seq} is the time to execute the same task on a sequential machine then the speedup is given by

$$S_p = \frac{T_{seq}}{T_{\max}} \tag{10}$$

Therefore, by minimizing T_w, the achievable speedup can be maximized. In the following we discuss such techniques, and in the next section we present the performance results for a parallel implementation of algorithms in the motion estimation system.

3.1.1. Uniform Partitioning

Data decomposition using uniform partitioning performs well as a load balancing strategy for input data independent tasks, because equally dividing the data distributes the computation equally among processors. If total input data size is D then total computation time to execute a task is $T = k \times D$, where k is determined by the computation at each input data point. For example, in convolution of an image with $m \times m$ kernel, $k = 2 \times m^2$ floating point operations. Hence, for an n node multiprocessor, the data decomposition methods to balance the computation is to make the granule size to

$$d_i = \frac{D}{n} \tag{11}$$

For data independent algorithms such a partitioning guarantees equal distribution of computation among processors. Therefore, if communication time can be minimized, then optimal performance can be obtained on a given multiprocessor.

3.1.2. Static

When computation is not uniformly distributed across the input domain, and is data dependent, uniform partitioning does not work well for load balancing. Normally, computation depends on significant data elements in a partition. Many vision algorithms exhibit this behavior. For example, in stereo match, hough transform etc., the computation is proportional to the number of features (edges) or significant pixels in a granule rather than on the granule size. Therefore, equal size granules do not guarantee load balanced partitioning because of the data dependent nature of the computation. In many such algorithms, the computation time for a granule (i), T_i, is proportional to a certain extent on the granule size (fixed overhead to process a granule), and to the number of significant data in a granule. That is,

$$T_i = A \times d_i + B \times f_i \tag{12}$$

where, d_i is the granule size, f_i is a measure of significant data in granule (i), and A and B are arbitrary constants which depend on the algorithm. The objective is to divide the computation among processors such that each processor receives equal measure of computation. One way to assign a granule to a processor is to compute the total measure of computation and partition is as follows:

$$T_i = \frac{\sum_{i=i}^{i=g} A \times d_i + B \times f_i}{n} \tag{13}$$

where, g is the total number of granules in the input domain (Note that the number of granules for the current task is n for an n processor system).

For example, consider computing hough transform of an edge image to detect line segments. If there exists a line whose normal distance from the origin is r, the normal makes an angle θ with the x-axis then if a point (x,y) lies on that line, the following Equation is satisfied.

$$r = x\cos\theta + y\sin\theta$$

r and θ are quantized for desired accuracy and then for each significant pixel (where there is an edge), r is computed for all quantized θ values. If two partitions of equal size contain different number of edge pixels, then the amount of computation will be different for the two partitions, despite them being equal in size. In fact, the computation is directly proportional to the number of edge pixels in a partition. One way to perform static load balancing is to decompose the input data such that each partition contains an equal number of edge pixels. The computation to recognize this partioning can be performed in the task in which edges are detected by keeping a count of the number of edges detected by a processor. Note that it is important to compute the statistics on the fly when edges are detected to guarantee low overhead. If the same statistics are gathered by sequentially scanning the input data then the overhead can be significant. Once a task is completed, the data can be reorganized such that the number of edges with each processor is in the interval $(\frac{Z_a}{n} - \delta\ ,\ \frac{Z_a}{n} + \delta)$, where Z_a is the total number of edges detected in the image, and δ is determined by the minimum granule size from fixed overhead considerations.

3.1.3. Weighted Static

When the computation in a granule not only depends on number of significant data points in the input domain, but it also depends on their spatial relationships, then data distribution needs to be taken into account as a measure of load to perform load balancing. For example, in stereo matching or time matching, not only does the computation depend on the number of zero crossings, but it also depends on their spatial distribution. If the zero crossings are densely spaced, then the computation will be more than that if the same number of zero crossings are sparsely distributed (refer to Figure 4). The reason is that if the zero crossings are densely packed, then more number of zero crossings need to be matched with each corresponding zero crossing in the other image, whereas less number of zero crossings need to be matched if they are sparsely distributed. Hence, the computation also depends on the spatial density (such as features/row). That is,

$$T_i = A \times d_i + B \times w_i \times d_i \tag{14}$$

where w_i is the feature dependent spatial density. For example, if the minimum granule size is a row of the input data then $w_i = r_i^{\beta}$, where r_i is the number of features in row i, and β is a parameter, $0 \le \beta \le 1$. $\beta = 0$ means that the computation is independent of how the features are distributed within a row. Therefore, to divide the computation equally among n processors, the following heuristic can be used.

$$T_i = \frac{\sum_{i=0}^{i=R} A \times d_i + B \times w_i \times d_i}{n} \tag{15}$$

where, R is the number of rows in the image. Note that the above heuristics approximate the load and do not exactly divide the computation among processors.

For example, in the stereo match computation, while partitioning the data among processors, a weight is assigned to each row as a function of number of features in the row. This weight represents the feature density. Note that using a row as the smallest granule avoids the communication overhead because search space for stereo matching is one dimensional, and therefore, if the granule boundary is one row then there is no need for communication.

3.1.4. Dynamic

Above three methods use the knowledge about the data when it is produced to perform load balancing for the next task. However, once decomposition is done, then the data is not reshuffled. Therefore, we

consider the above methods as knowledged based static load balancing schemes. In the dynamic scheme, the data is decomposed into finer granules such that the number of tasks, (that is number of independent granules) M, is much larger than the number of processors.

At execution time the processors are assigned these tasks dynamically by a designated scheduler from a task queue containing these tasks. Processors are assigned new tasks as they finish their previously assigned tasks, if there are more tasks left to be assigned. However, the knowledge obtained from the previous step can be used again to anticipate the completion of a task, in order to assign a new task to a processor. That is, the task assignment can be pipelined, thereby reducing the overhead of dynamic assignment.

The following procedure illustrates the dynamic assignment of tasks onto the processor. The pseudo code essentially illustrates what the scheduler does in order to perform dynamic load balancing. The number of tasks (max_tasks) are determined during the execution of the preceding step in the system, and the task_queue contains all the tasks including the computational information associated with each task. Initially, the scheduler assigns few tasks to each processor. The number of tasks to be assigned initially is a parameter (pipe_line_no). If this parameter is 1, it implies that there is no anticipatory scheduling. In other words, a processor is assinged a new task only when it finishes the task it is currently executing. A task is assigned to a processor only if the task contains significant computation. For example, in stereo match, if a task's data does not contain any zero crossings, then the task can be discarded because it is not going to produce any useful information anyway. In a blind scheme, where little is known about a task, the task will be assigned, which is an overhead, and can be avoided by using the knowledge obtained from the previous steps. Whenever a processor P_i completes the current task, it sends a *compl_msg* to the scheduler which assigns P_i a new task if the task_queue is not empty. Once the task_queue becomes empty, the scheduler sends a *term_msg* (terminate message) to all the processors. Upon receiving a *term_msg* from the scheduler, processors complete the remaining tasks in their task_queues, and sends a *term_msg* to the scheduler, terminating the computation. Note that by using the pipe_line_no, anticipatory dynamic scheduling can be performed, and a processor need not be idle when a new task is being assigned. By using this parameter, the amount of initial static assignment, and dynamic assignment can be controlled. Figure 17 shows the partitioning for the above described strategies for stereo match algorithm.

Dynamic Scheduling of Tasks

```
/*Initial Assignment*/

1.    curr_task = 0;
2.    for j = 1 to j <= pipe_line_no do
3.          for i = 1 to i = num_proc do
4.                if comp(task_queue(curr_task)) > 0
5.                      schedule curr_task at proc. P_i;
6.                      curr_task = curr_task+1;
7.                else
8.                      curr_task = curr_task+1;
9.                      go to 4.
10.               end_if
11.         end_for
12.   end_for

      /*Scheduling*/

13.   done = false; k = num_proc;
14.   while not done do
15.         wait for msg from a processor;
16.         receive msg;
17.         if ( msg = compl_msg )
18.               P_i = sender processor;
19.               if curr_task < max_tasks
20.                     if comp(task_queue(curr_task)) > 0
21.                           schedule curr_task at proc. P_i;
22.                           curr_task = curr_task+1;
23.
24.                     else
25.                           curr_task = curr_task+1;
26.                           go to 19.
27.               else
28.                     send term_msg to P_i.
29.         else if ( msg = term_msg)
30.               k = k - 1;
31.               if (k <= 0)
32.                     done = true.
```

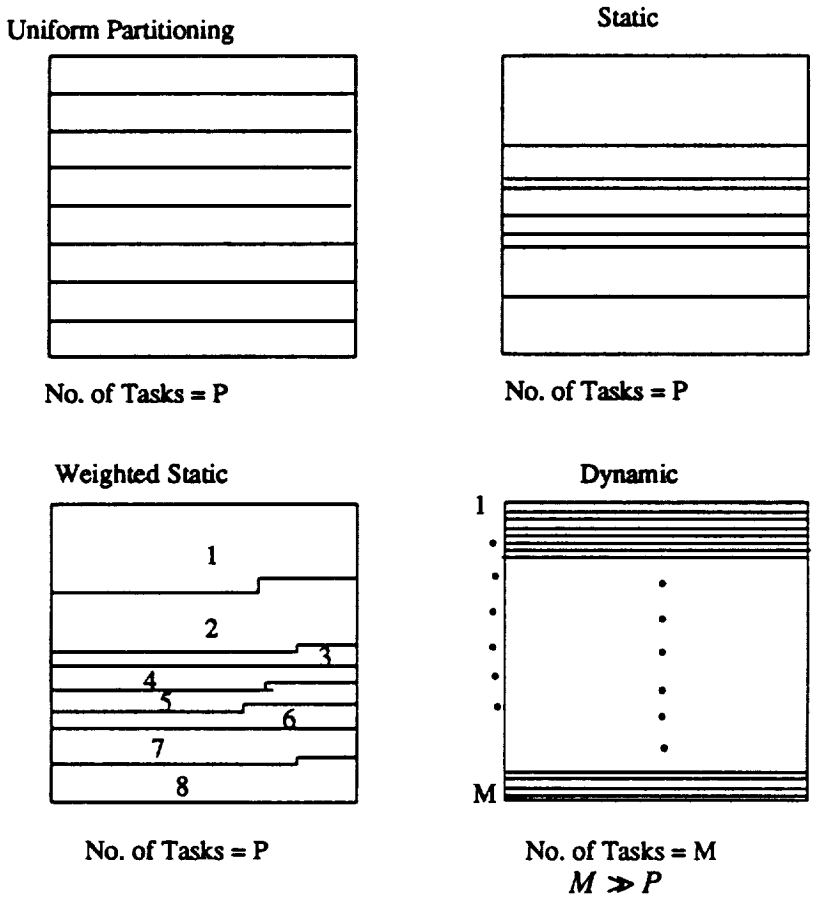


Figure 17 : Load balancing strategies

4. Parallel Implementation and Performance Evaluation

This section presents a parallel implementation of the algorithms that are part of motion estimation system and describes the performance of the algorithms and load balancing strategies. The algorithms were implemented and evaluated on a hypercube multiprocessor.

4.1. Hypercube Multiprocessor

A hypercube multiprocessor system of size P has P processors, where P is an integral power of 2. P processors are indexed by the integers 0,...,P-1 and the following criteria is satisfied. If the processor numbers are represented by $\log_2(P)$ bits then two processors are connected by communication links if and only if their bit representation differs by exactly one bit. Therefore, each processor is connected to $\log_2(P)$ processors with direct communication links. Diameter of the hypercube of size P is $\log_2(P)$ (diameter is the maximum distance between any two nodes). Figure 18 shows a 4-dimensional hypercube

multiprocessor.

A typical commercially available hypercube multiprocessor system consists of a host processor and node processors. The host processor serves as the cube manager, provides interface with the external environment, provides input-output of data and program. We used Intel ipsc/2 hypercube multiprocessor consisting of 16 nodes. Each node consists of an Intel 80386 processor, Intel 80387 co-processor, 4 mega-byte memory, and a communication module.

4.2. Feature Extraction

Features used for stereo match algorithms are the zero crossings of the convolution of the image with Laplacian. Zero crossing computation involves 2-D convolution and extraction of zero crossings from the convolved image. Since convolution is a data independent algorithm uniform partitioning is sufficient to evenly distribute the computation. The mapping is a division of $N \times N$ image onto P processors. Each processor computes the zero crossings of share of N^2/P pixels (Equation 11). Data division onto the processors is done along the rows. This mapping reduces communication to only in one direction. The reason is that 2-D convolution can be broken into two 1-D convolution [7]. This not only reduces the computation from W^2 sum of products operations per pixel to $2 \times W$ sum of product operations per pixel (W is the convolution mask window size), but also reduces the communication requirements in a parallel implementation if the data partitioning is done along the rows. There is no need for communication when convolution is performed along the rows.

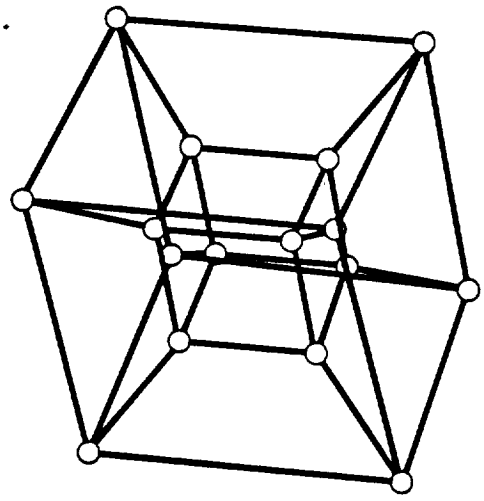


Figure 18 : A 4-Dimensional hypercube

Table 1 shows the performance results for the above implementation for an image of size 256×256 and convolution window of size 20×20. First column shows the number of processors in the cube(P). Second column represents the total processing time (t_{proc}) for convolution. Column 3 shows the number of bytes communicated by a processor to the neighboring processor, and column 4 shows the corresponding communication time which is small compared to the computation time. The second half of the table shows the computation time for extracting zero crossings from the convolved image. Corresponding speedups are also shown.

It can be observed that almost linear speedup is obtained for convolution. Two factors which contribute toward this result are that communication overhead is relatively small, and communication is constant as the number of processors increases. However, the speedup obtained in the elapsed time, which includes the program and data load time also, is sub-linear due to the following reason. The hypercube multiprocessor's host does not have a broadcast capability, and therefore, the overhead of loading the program increases linearly with the number of processors. However, data load time increment with the increase in the number of processors is comparatively small because amount of data to be loaded to one processor decreases as the number of processors increases. The only increment in data load time results from the number of communication setups from the host to the node processors, which increases linearly

Table 1 : Performance for feature extraction (zero crossings)

Computation for Convolution and Zero Crossings							
	Convolution Window Size = 20x20						
No. Proc.	Conv. Comp. Time(sec.)	Conv. Comm. Bytes	Conv. Comm. Time(ms.)	Conv. Total Time(sec.)	Conv. Speed Up	ZC Comp. Time(sec.)	ZC Speed Up
1	109.0	0	0	109.0	1	6.47	1
2	54.76	2816	13	54.78	1.98	3.23	1.99
4	27.51	5632	36	27.55	3.95	1.66	3.89
8	13.88	5632	36	13.92	7.83	0.85	7.60
16	7.07	5632	36	7.11	15.33	0.42	15.25

Feature Extraction Performance (Elapsed Time)		
No. Proc.	Elapsed Time(sec.)	Speed up
1	116.2	1
2	58.8	1.97
4	30.1	3.86
8	16.1	7.22
16	9.6	12.1

with the number of processors.

4.3. Matching Features

This task involves matching features in stereo pair of images. As discussed in section 2, the epipolar constraint limits the search for a match in the corresponding image to only in horizontal direction, i.e., along the rows in the zero crossings of the image. Thus data partioning along the rows for parallel implementation results in no communication between node processors as long as each partition contains an integral number of rows.

The computation involved in stereo matching algorithm is data dependent. The computation varies across the image because it depends on the number of zero crossings, distribution of zero crossing across the image, and distribution of zero crossings along the epipolar lines. Therefore, partioning the data uniformly among the processors (i.e. assign each processor equal number of rows) may not yield expected speedups and processor utilization. A processor which has very few zero crossings, and sparsely distributed zero crossings will be under utilized, whereas a processor with a large number of zero crossings, and densely distributed zero crossings will become a bottleneck.

We used uniform partitioning, static load balancing, weighted static and dynamic load balancing schemes to decompose the computation on the multiprocessor. Static load balancing can be achieved by keeping a count of the zero crossings with each processor when the previous task (feature extraction) is executed. At the completion of the task, the data is reorganized using this information, and using the techniques described in the previous section.

Figure 19 shows the distribution of the computation times for 8 processor case. The X-axis shows the processor number, and the Y-axis shows the computation time for each scheme. As we can observe, uniform partitioning does not perform well at all because the variation in computation time is tremendous, and therefore, performance gains are minimal. The static load balancing scheme (shown as dashed bars) performs much better than uniform partitioning, but variation in computation times is still significant because the computation also depends on the distribution of zero crossings. The weighted static scheme performs better than static, and further reduces the variation in computation times. Note that these schemes only measure the load approximately, and therefore, will not divide the computation exactly uniformly. Furthermore, minimum granularity is a row boundary in order to avoid communication between

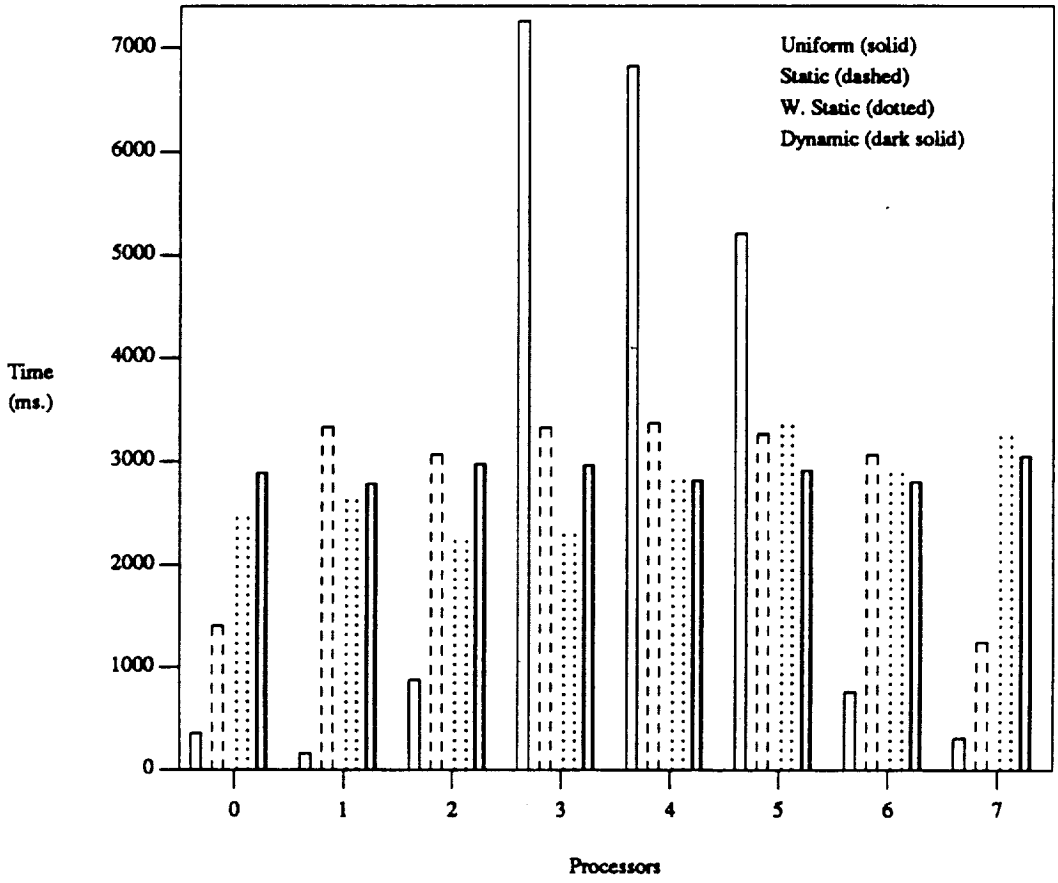


Figure 19 : Distribution of computation times for stereo match (P=8)

processors. Finally, for 8 processor case, dynamic scheme performs the best. Table 2 summarizes the distribution for the 8 processor case. The Table shows the computation time, variation ratio, and improvement ratio for each processor under all four methods. For example, the variation ratio is 44.25 for uniform partitioning, is 2.71 for static load balancing, is 1.50 for weighted static, and is 1.09 for dynamic load balancing. Improvement ratio is the ratio of speedup obtained with load balancing to that of uniform partitioning. The computation times shown include all the overhead of load balancing schemes. Figure 20 depicts the speedup graph for varying size of multiprocessor from 1 processor to 16. We observe that uniform partitioning does not provide any significant gains in speedup as the number of processors increases. Dynamic scheme performs the best among all the schemes, and the two static scheme perform comparably with the dynamic scheme. We believe that as the number of processors is increased, the two static schemes will move even closer to dynamic scheme, or even perform better than the dynamic scheme, because for a larger multiprocessors, the overhead of dynamic scheme will be greater.

Table 2 : Distribution of computation times for stereo match

Computation Time Distribution for Stereo Match (P=8)				
Proc. No.	Uniform Partitioning	Static	Static Weighted	Dynamic
	Time (ms.)	Time (ms.)	Time (ms.)	Time (ms.)
0	364	1402	2439	2890
1	164	3333	2606	2786
2	878	3066	2219	2980
3	7258	3327	2277	2967
4	6827	3371	2798	2818
5	5207	3269	3328	2913
6	762	3063	2864	2803
7	312	1243	3223	3051
Max.	7258	3371	3328	3051
Min.	164	1243	2219	2786
Variation ratio	44.25	2.71	1.50	1.09
Improvement ratio	1	2.15	2.19	2.38

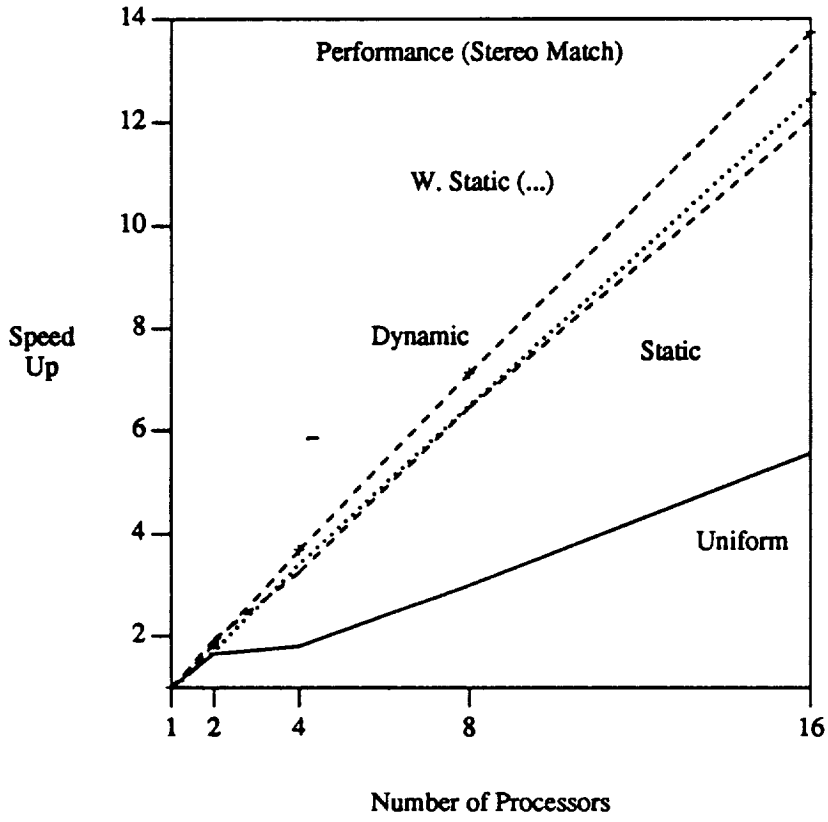


Figure 20 : Speedups for stereo match computation

4.4. Time Match

The computation in time match algorithm is similar to that in stereo match except the search space is two-dimensional, and the input to the algorithm is stereo match output. Other difference is that the number of significant points in the input data is much smaller than that in stereo match, because a great deal of input points get eliminated in stereo match. Table 3 shows the distribution of the computation times for the 16 processor case. We only present uniform partitioning and static load balancing cases. The most important observation is that uniform partitioning performs worse than that in the case of stereo match, and static load balancing performs better.

The Table shows how the measure of computation (number of zero crossings left from stereo match step) is divided among the processors in the two cases. It is clear that the number of zero crossings are very evenly distributed (within the minimum granule of one row constraint) in the static case, whereas they are lumped with a few processors in the uniform partitioning case. Figure 21 shows the speedup graphs for the two schemes for a range of multiprocessor sizes. The speedup gains for the load balanced case is very

Table 3 : Distribution of computation time for time match step

Computation for Time Match (Proc. = 16)						
Proc. No.	Uniform Partitioning			With Load Balancing		
	Matching (Sec.)	Total (Sec.)	No. Zcs	Matching (Sec.)	Total (Sec.)	No. Zcs
0	0.14	0.22	3	9.35	10.00	47
1	0.03	0.14	2	12.38	12.55	50
2	0.02	0.13	0	13.12	13.21	53
3	0.02	0.13	0	14.23	14.27	43
4	0.02	0.13	0	11.88	11.91	45
5	3.61	3.72	21	10.93	10.95	44
6	13.45	13.56	55	12.82	12.85	53
7	5.09	5.20	20	12.16	12.19	51
8	26.65	26.76	93	11.41	11.44	45
9	45.85	45.97	182	10.63	10.65	40
10	73.82	73.93	259	13.89	13.91	50
11	27.20	27.32	121	13.69	13.71	44
12	0.31	0.42	3	15.07	15.09	43
13	0.11	0.22	1	15.70	15.72	56
14	0.42	0.53	4	14.36	14.39	56
15	0.08	0.10	0	5.21	5.68	43
	Max. time(sec.)	Min. time(sec.)	Variation ratio	Speed up	Improvement ratio	
Uniform	73.82	0.10	738	2.69		
Balanced	15.72	5.68	2.76	12.63	4.7	

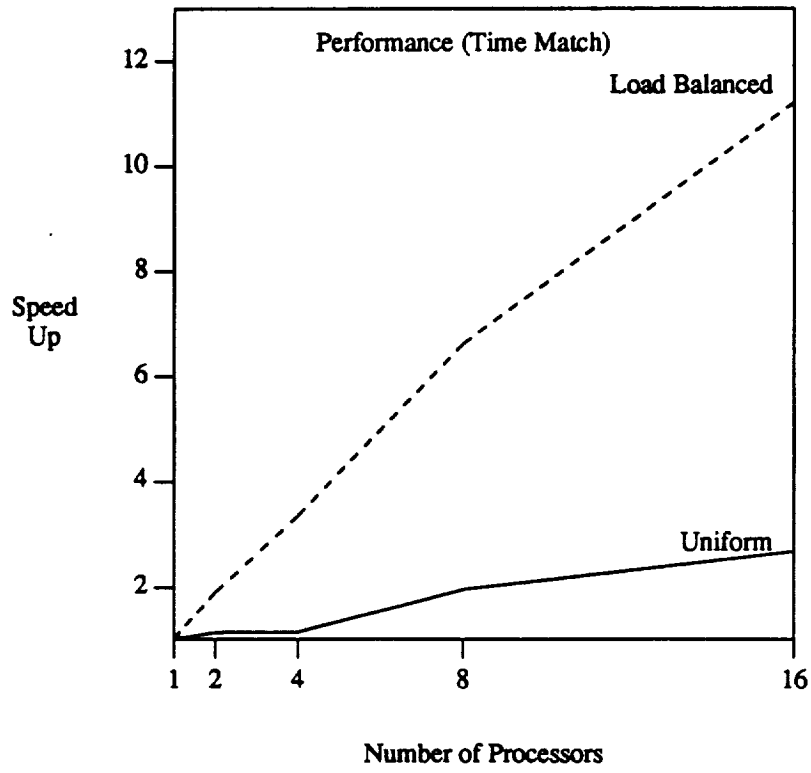


Figure 21 : Speedup for time match step

significant over the uniform partitioning case. We computed the overhead of performing knowledge based static load balancing, and the overhead was 3 ms., which is negligible compared to the computation time, and the performance gains are significant.

4.5. Second Stereo Match

This step involves stereo match computation for features from images at time instant t_{i+1} after time point correspondence is established between images at time t_i and t_{i+1}. The matching is similar to that in first stereo match except that it need to be done only at those points at which time correspondence has already been established. Consequently, the number of features to be matched are much less than that in the first computation, and hence, the importance of load balancing is further increased. Figure 22 depicts the distribution of computation times for the second stereo match step. The three load balancing algorithms used in this case are Uniform Partitioning, Static and Dynamic. We observe from the Figure that uniform partitioning does not perform well compared to the other two schemes. The variation in computation time is significant, and the static and dynamic schemes perform comparably.

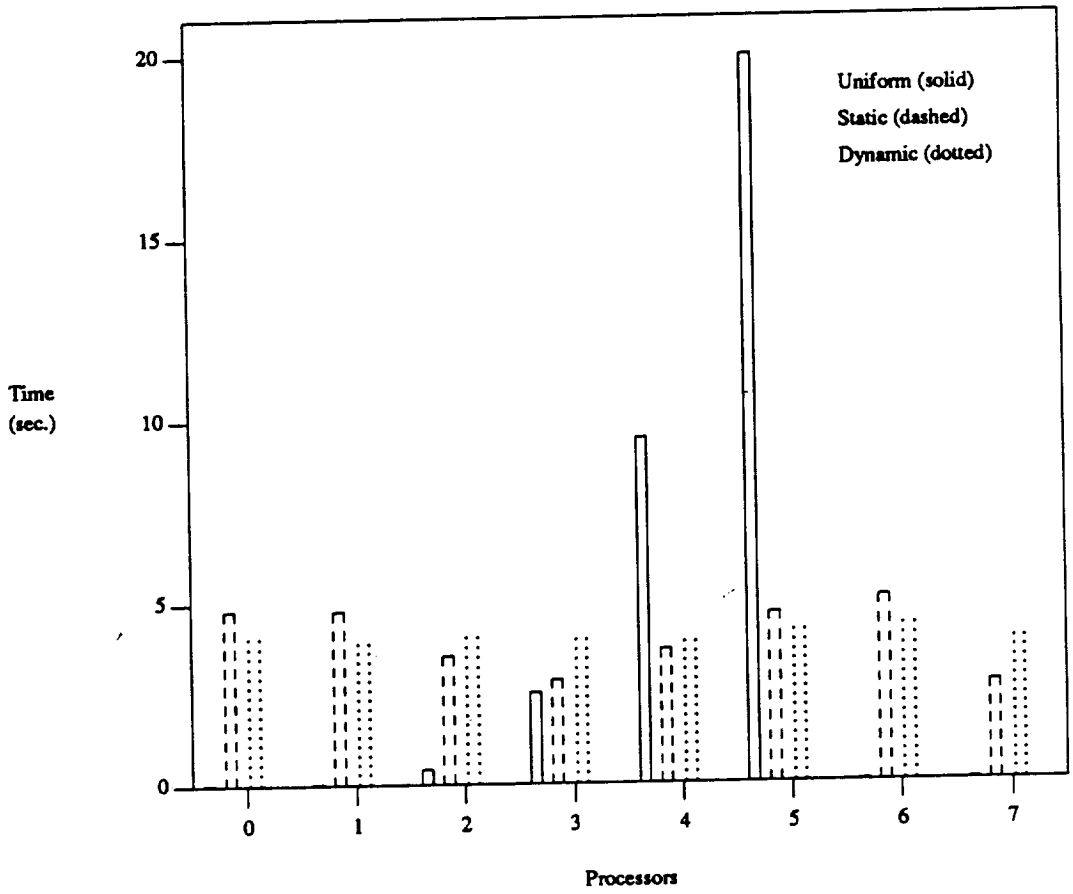


Figure 22 : Distribution of computation times for second stereo match (P=8)

Figure 23 presents the speedups for the same algorithm for various multiprocessor sizes. The Figure shows that the gains from these load balancing schemes are very significant over uniform partitioning. One important observation can be made by comparing results in Figures 20 and 23. Note that the performance of uniform partitioning in the second stereo match is much worse than that in the first stereo match. For example, for 16 processor case, the speedup in the first case is 5.55, whereas for the same multiprocessor size, the speedup is only approximately 2.3 in the second case. Therefore, as the computation progresses from one step to the next in a vision system, the gains of these load balancing schemes become increasingly significant.

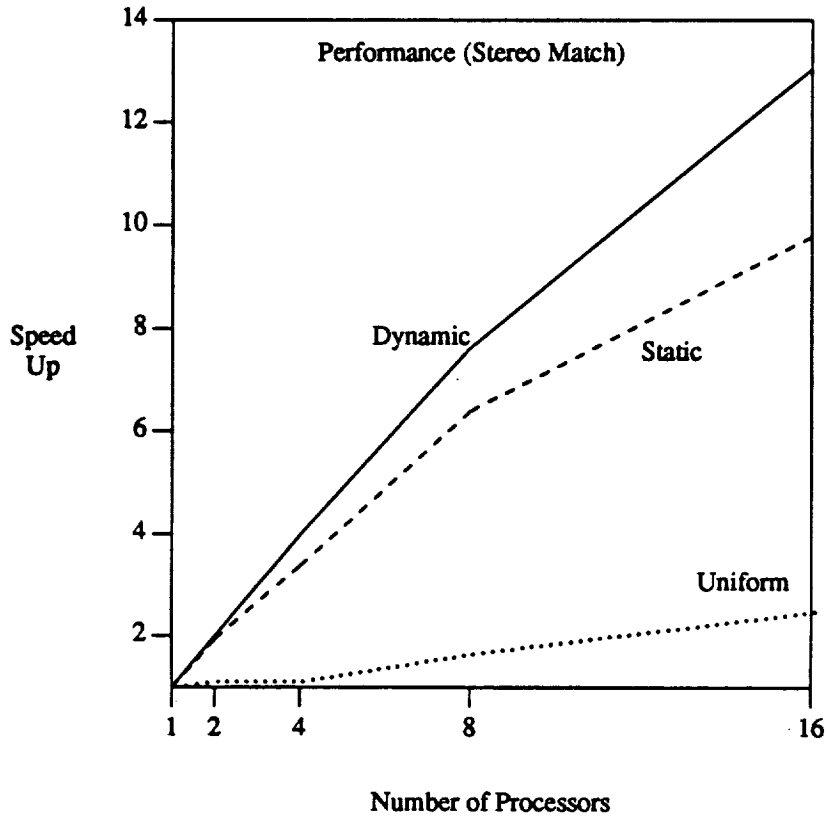


Figure 23 : Speedups for second stereo match

4.6. Summary of Results

In summary, the following important observations can be made from the results presented in this section. First, the improvement in performance (such as utilization and speedup) itself increases using the load balancing schemes as the number of processors increases. Therefore, performance gains are expected to be higher for larger multiprocessors. Second, in an integrated environment, the overheads of such methods are small because measure of load can be computed at run time as a bi-product of the current task. Finally, though we showed the performance results of the implementation on the hypercube multiprocessor, these methods can be applied when algorithms are mapped on any medium to large grain multiprocessor system, because these techniques are independent of the underlying multiprocessor architecture.

Consider the overall performance gains for the entire system. As the computation progresses from one step to the next, uniform partitioning performs worse because the data points reduce, but the computation at each point increases. Hence, the gains of using parallel processing are minimal. However, the load

balancing techniques recognize the data distribution at each step, and the data is decomposed using the distribution. Therefore, performance gains are expected to improve as the computation progresses in an integrated environment. For example, consider zero crossing, stereo match, time match, and second stereo match steps. In zero crossing computation, uniform partitioning performs well and load is balanced. Hence, the improvement ratio is 1. For stereo match the improvement of static over uniform partitioning is 2.15 for 8 processor case, and is 2.22 for 16 processor case. Similarly, for time match step, the improvement of static load balancing for 8 processor case is 3.38, and for 16 processor case, it is 4.2. Therefore, the improvement in performance itself increases as the number of processors increases as well as when the computation progresses in from one step to the next in a vision system.

5. Concluding Remarks

The first part of this paper presented algorithms for a motion estimation system with an emphasis on obtaining point correspondences. The algorithms are non-iterative, and obtain matched features points among stereo images at any two consecutive time instants. The system consists of the following steps: 1) extraction of features, 2) stereo match of images in one time instant, 3) time match of images from different time instants, 4) stereo match to compute final unambiguous points and, 5) computation of motion parameters. Since, zero crossings points are used as the features for matching, there are only about 7% (as shown in Table 4) of the total number of points (65536) involved in matching process. Consequently, the algorithms save considerable amount of computation in solving the correspondence problem. Table 5 shows the number of unambiguous matched point pairs at various stages. The number of matched point pairs among the images is 262 which seems enough for motion estimation. Currently, we are trying different motion parameter computation algorithms on the matched point pairs.

Second, we presented techniques to perform efficient data decomposition and load balancing for vision systems, for medium to large grain parallelism. Two important characteristics of these techniques are that they are general enough to apply to many such systems, and that they use statistics and knowledge from the execution of a task to perform data decomposition and load balancing for the next task in the system. The advantages of such schemes are as follows. First, these techniques use characteristics of the tasks and the data, and therefore, work well no matter how the data changes. Second, many vision systems consist of similar tasks, and exhibit similar computation flow, and therefore, these techniques can be used

Table 4 : Number of feature points in each image

Image	no. of z. c. pattern pts.
I7	4185
r7	4348
I8	4255
r8	4294

total number of points in each image = 65536

Table 5 : Number of unambiguous matched point pairs at various matching stages

	stereo matching I7, r7	time matching I7, I8	stereo matching I8, r8	among I7, r7, I8, r8
total unambiguous matches	625	373	262	262

in any system.

Finally, the performance of the proposed techniques was evaluated by using a parallel implementation of the motion estimation system algorithms on a hypercube multiprocessor system. The results show that using uniform partitioning without considering the computations involved, parallel processing does not provide significant performance improvements over sequential processing. Furthermore, by applying the proposed data decomposition and load balancing techniques significant performance gains (as much as 6 fold) can be obtained over uniform partitioning.

REFERENCES

[1] C. Weems, A. Hanson, E. Riseman, and A. Rosenfeld, "An integrated image understanding benchmark: recognition of a 2 1/2 D mobile," in *International Conference on Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 1988.

[2] Alok N. Choudhary, "Parallel architectures and parallel algorithms for integrated vision systems," in *Ph.D. Thesis*, University of Illinois, Urbana-Champaign, Agust 1989.

[3] Alok Choudhary and Janak Patel, "A parallel processing architecture for integrated vision systems," in *17th Annual International Conference on Parallel Processing*, St. Charles, IL, pp. 383-388, August 1988.

[4] Mun K. Leung and Thomas S. Huang, "Point matching in a time sequence of stereo image pairs," in *Tech. Rep., CSL, University of Illinois*, Urbana-Champaign, 1987.

[5] Alok N. Choudhary, Subhodev Das, Narendra Ahuja, and Janak H. Patel, "Surface reconstruction from stereo images : an implementation on a hypercube multiprocessor," in *The Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, Monterey, CA, March 1989.

[6] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-sqaure fitting of two 3-D point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, pp. 698-700, September 1987.

[7] A. Huertas and G. Medioni, "Detection of intensity changes with subpixel accuracy using Laplacian-Gaussian masks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, pp. 651-664, September 1986.

[8] Y. C. Kim and J. K. Aggarwal, "Positioning 3-D objects using stereo images," *Computer and Vision Research Center, The University of Texas at Austin.*

NASA
FORMAL
REPORT

FFNo 665 Aug 65