N90-16451

PARALLEL ALGORITHM FOR DETERMINING MOTION VECTORS IN ICE FLOE IMAGES BY MATCHING EDGE FEATURES

M. Manohar, H. K. Ramapriyan and J. P. Strong

NASA/ Goddard Space Flight Center Space Data and Computing Division, Information Systems Facility, Code 636, Greenbelt, Maryland 20771

ABSTRACT

A parallel algorithm is described to determine motion vectors of ice floes using time sequences of images of the Arctic ocean obtained from the Synthetic Aperture Radar (SAR) instrument flown on-board the Seasat spacecraft. Time intervals between two successive images of a given region can be as much as three days. During this period, large translations and rotations of ice floes can occur. Therefore, conventional local correlation techniques which perform searches in a small neighborhood to detect translated features have a very small chance of success. To account for large translations and rotations, it is necessary to perform large area searches in a three dimensional space (two translational and one rotational). This makes conventional correlation techniques computationally intensive even on a high-speed parallel computer such as the Massively Parallel Processor (MPP). In this paper we describe a parallel algorithm which is implemented on the MPP for locating corresponding objects based on their translationally and rotationally invariant features. The algorithm first approximates the edges in the images by polygons or sets of connected straight-line segments. Each such "edge structure" is then reduced to a "seed point". Associated with each seed point are the descriptions (lengths, orientations and sequence numbers) of the lines constituting the corresponding edge structure. A parallel matching algorithm is used to match packed arrays of such descriptions to identify corresponding seed points in the two images. The matching algorithm is designed such that fragmentation and merging of ice floes are taken into account by accepting partial matches. The technique has been demonstrated to work on synthetic test patterns and real image pairs from Seasat in times ranging from .5 to 0.7 seconds for 128 x 128 images.

INTRODUCTION

Sequential images of ice floes in the Arctic ocean were obtained from the Synthetic Aperture Radar (SAR) flown on-board the Seasat spacecraft in 1978. Using time sequences of these images, it has been shown in the literature that it is possible to map ice motion. The approach taken is to match recognizable features in the ice field which are imaged from two successive orbits. The matching procedures have been traditionally manual and time consuming. In order to perform this task routinely on a large number of images, it is necessary to develop automated analysis techniques. Recently, several automated techniques of estimating ice motion using cross correlations have been proposed (for example, [1-3]). Collins [3] posed the problem of finding a field of displacements between two successive images as an estimation problem. The typical time interval between two images of a given region is of the order of three days. During this period, large differential translations and rotations of ice floes can occur. Therefore, conventional local correlation techniques which perform searches in small local neighborhoods for displaced features have a very small chance of success. To account for large translations and rotations, it is necessary to perform large area searches in a three dimensional space (one rotational and two translational dimensions). These factors make correlation techniques computationally intensive even on a high-speed parallel computer such as the Massively Parallel Processor (MPP). Additional problems specific to ice floe images are fragmentation and merging during the time interval between the images. This requires approximate matches of ice floes from one image to the parts of larger ice floes from the other images. This problem is referred to as segment matching in the literature [4].

The work reported in this paper is an effort to automate ice floe matching in computationally feasible times (a few seconds for a pair of 512 x 512 images) using the MPP. In our approach the images are abstracted as line models of the boundaries of dominant objects (ice floes) in the image, and these models are matched using parallel matching techniques. The boundaries of the dominant objects are extracted by edge detection algorithms and the edges are segmented into set of lines by fitting polygons or connected sets of straight lines to the edge data. Such polygons or connected sets of straight lines will be referred to as *edge structures*. Now the problem is to match corresponding edge structures in the two images. Edge structure models have been used in the literature for matching cloud images [5] and terrain scenes [6]. Both these techniques are essentially sequential and are not considered segment matching. Davis [4] uses a relaxation technique for segment matching of edge structures. Initially, figures of merit are assigned to the matches between pairs of angles on two edge structures. Relaxation methods are then used

to find acceptable combinations of these matches. This method is also sequential and is not a practical solution for images containing thousands of edge structures.

THE ALGORITHM

The algorithm presented in this paper consists of the following steps, each of which is implemented in parallel. The edges of ice floe images are obtained by a suitable edge detection algorithm. The edges obtained are further subjected to some preprocessing such as thinning and eliminating isolated edge points. A connected component labeling algorithm [7] is applied to the edge map to obtain a label array, LBL. This algorithm locates a seed point in each connected set of edge points in the edge map and assigns the address of the seed point to all edge points in the connected set. Next, each edge is decomposed into a set of straight line segments. This is accomplished by detecting the corner points in the edge map. The corner point detection algorithm examines a local window (typically 7 x 7) and fits a straight line passing through its center. The fitting error, equal to the sum of the perpendicular distances from the edge points of the window to the fitted line is computed. The locations at which this error function has local maxima are identified as corner points. By assigning 0's to all corner points in the edge map the connected sets of edge points are separated into straight line segments to obtain a segmented edge map. Now, the length of each straight line segment is computed by shrinking the segmented edge map and counting the number of shrink operations each segment undergoes. The lengths so computed are stored in an array LEN, with the mid-points of each line segment containing its length and all other points containing 0. The orientation of each line segment is computed by applying the Hough transform to local windows (typically 5 x 5) surrounding its mid-point. This information is stored in an array, DIR, at addresses corresponding to the mid-points of the line segments. Next, sequence numbers are assigned to all line segments within each connected component indicating the order in which they are connected. This algorithm processes the edge map and LBL arrays to produce sequence numbers. This information is stored in an array, SEQ. Now, the quadruple (LBL,SEQ,DIR, LEN) provides a complete description of the edge structures in the image. These quadruples are sorted with LBL and SEQ as the primary and secondary keys, respectively, using a bitonic sorting algorithm [12]. This brings all data values of the arrays together into adjacent processing elements. Note that the arrays of sorted attributes is significantly smaller than the original images. This can make the matching more efficient, especially for images significantly larger than the n x n processing array if attribute records are packed by processing n x n segments of the images. These sorted arrays obtained from two images are then matched using a combination of global and local correlations. In the following sections we describe each of these steps in more detail.

Preprocessing:

The preprocessing step consists of obtaining the boundary of the dominant objects of the ice floe images and applying thinning and eliminating isolated edge points. In the present work, various edge detection techniques such as Marr-Hildreth's zero crossing detection [8], Canny's algorithm [9], and Spatially Constrained Clustering (SCC) [Tilton, 10] were applied to the ice floe images. Both Marr-Hildreth's and Canny's algorithms gave unacceptable results. The zero crossings obtained by convolving the Laplacian of Guassian with the image combined the boundaries of the dominant objects with several other details and so it was difficult to delineate the boundaries from the zero crossings. Larger filter sizes could solve this problem to some extent but the edge location accuracy becomes poor. The Canny's algorithm also has the same problem for the ice floe images. The smaller filter gives unnecessary edges (due to noise and fine texture) and large filter sizes affect the shape of the boundary. Tilton's SCC algorithm [10], which grows regions based on a "best pair merging" criterion, performed better than the other two in delineating boundaries of dominant objects. The edge focussing algorithm by Bergholm [11] performed best for these images. This edge focussing algorithm is as follows.

1. Initialize a mask (of the size of the image) to 1's.

2. Detect major edges using Canny's algorithm with large size Guassian filters (e.g., $\sigma = 7.0$) and an appropriate threshold (e.g., 0.1) for the image. Accept edges only at locations where mask has values 1 as true edge.

3. Dilate the edges obtained in step 2 by one pixel in each direction and generate a mask which is 1 at all the dilated edge locations.

4. Now decrease sigma of the filter by 0.5. (It is shown analytically in [11] that change of sigma by 0.5 can displace an edge location by at most one pixel on either side of its previous location). Repeat steps 2 and 3 until the filter size is 0.5.

In order to speed up convolution the filter size also can be reduced in steps of two for every iteration starting with a size of 15 x 15.

This algorithm takes about 200 ms for 128 x 128 images on the MPP to perform 7 iterations with window sizes ranging from 15 x 15 to 5 x 5. The results of the edge detection algorithm are shown in Fig. 2(a - b) for both the images of the given pair Fig.1(a-b).

The edges obtained using the above algorithm are one pixel wide, but they are 4-connected for inclined (other than horizontal and vertical) lines. The subsequent processes involved require inclined edges to be 8-connected. A thinning algorithm is used to reduce such inclined 4-connected edges to 8-connected edges.

The thinning algorithm examines a 3 x 3 neighborhood and replaces the central pixel by 0 under the following conditions:

0 0 1	x 1 x	0 0 x
0 1 1	0 1 1	0 1 1
0 0 1	x 1 x	x 1 x

In the above configurations 0 denotes the absence of an edge point, 1, the presence and x, a don't care condition. The other configuration of masks can be obtained by rotating the above masks by 90 degrees. The elimination of isolated edge points is straightforward.

Connected Component Labeling:

The connected component labeling algorithm labels each of the connected edges with a label that is equal to the array address of the seed point. The algorithm is discussed in detail in [7]. This algorithm is based on parallel shrinking and expansion of binary patterns and requires about 30 ms of MPP time for labeling the edges. The time required is data dependent and is proportional to the length of the within-component-path of the longest component. The connected component labelling reduces all connected patterns, both open and closed, to single points called seed points. The connected component labels obtained from the edge maps are shown in Fig. 3(a-b). The value of the label at a pixel is coded as its grey level so that all pixels belonging to a given connected component are displayed with the same grey level value.

Edge Decomposition:

Each connected component identified above is decomposed into a set of straight line segments by first identifying the corner points. The edge points between a pair of corner points are assumed to form a straight line. Corner points are edge points where the line direction changes significantly. The algorithm fits a straight line at every edge point to the connected set of edge points in a local neighborhood of size w x w (w = 7, typically). The fitting error (which is sum of perpendiculars from the edge points onto the fitted line) is computed. The local peaks in the fitting error function correspond to the corner pixels. The fitting error function is derived as follows.

Let $\alpha x + \beta = 0$ be the equation of the straight line passing through the origin of a local coordinate system to be fitted to the connected set of edge points in a w x w local neighborhood.

The error term, ε, is sum of perpendiculars from all connected edges of the local window which is given by

$$\varepsilon = \sum_i (\alpha x_i + y_i)^2 \qquad (1)$$

The value of a resulting in minimum ε is obtained by differentiating error, ε with respect to α and equating it to zero.

$$\alpha = -\sum_i x_i y_i \; / \; x_i^2 \qquad (2)$$

The minimum error for the best fit, ε_m can be obtained by substituting a from equation 2 in equation 1.

$$\varepsilon_m = \sum_i y_i^2 - (\sum_i x_i y_i)^2 / \sum_i x_i^2 \qquad (3)$$

High values of the error term ε_m indicate corners because a single line cannot be fitted to the given set of edge points. Therefore, local peaks of ε_m correspond to the corner points. The corner points detected in both images are shown in Fig. 4(a - b) superimposed over the corresponding edge images.

Length Computation :

The line segment lengths are computed by applying an 8-connected shrinking algorithm [7] repeatedly on the edge map wherein corner points are replaced with 0's. The number of shrinking operations required to reduce each line segment to a single point corresponds to the length of the line segment. Thus the lengths of the line segments are stored at their mid- points (called *line-seeds*). These points are the locations where all information needed for matching (such as label, direction, and length) about the line segments is stored. The length of the different edge segments are shown in Fig. 5(a-b) for two images. For displaying length as gray level image its value is propagated throughout the edge segment. The algorithm however, does not require this propagation.

Direction Computation :

The direction of each line segment is computed at the line-seeds using a localized version of the Hough transform. At every line-seed in the edge map a local neighborhood of size w x w is examined. For every point in the neighborhood with the same label as the line-seed, the angle subtended by the line joining it and the line-seed is computed. A 32-bin histogram of these angles over the w x w window is computed. This corresponds to an angular resolution of less than 6 degrees. The slope corresponding to the peak of the histogram is the direction of the edge segment. The direction of different edge segments are shown in Fig. 6(a-b) for two images.

Sequence Computation :

An ordered set of direction and length measurements corresponding to a sequence of connected line segments constituting an edge structure describes the edge structure completely. The order of occurrence of line segments is essential

for this description to be unique. Thus attaching sequence numbers to all line segments of the edge structures is an important step of the algorithm. The sequence numbers are attached to all edge points constituting the edge structure and the numbers at the location of line-seeds are retained as line attributes. The sequence numbers are attached to all points of the polygon boundary starting from seed points for closed polygons. The process is slightly different for open edge structures and will be discussed later. Initially a sequence number of 1 is attached to all seed points (starting points) of the polygons. Now a 5 x 5 neighborhood centered at one of the immediate neighbors of starting point is considered. This point is given the next sequence number provided none of the other seven neighbors of the starting point has already been given this number. Then the position of the starting point is shifted to the current pixel where sequence number is assigned. This is repeated until no more assignments are possible. This algorithm is sequential along perimeter of a given polygon, but operates in parallel on all polygons.

For open edge structures the seed points are the mid-points (rather than end points), so they are not suitable as starting points for sequence generation. One of the two end points should be considered as a starting point. To locate the starting point for all open edge structures in parallel, we proceed as follows. The line ends are detected by examining 3x3 windows and counting number of edge pixels surrounding the central edge pixel. If this number is equal to 1 then the central pixel is an end point. This is valid for thin edges where inclined edges are 8-connected (but not 4-connected). Then the sequencing algorithm is applied from the seed points as in the case of closed polygons. The sequencing algorithm terminates at one of the end points. Now, it is not difficult to locate unique end points in parallel. The points where the sequence array has a value greater than 1 and the line-end array has a value of 1 are the starting points for open edge structures.

Thus the sequence computation algorithm treats closed and open edge structures separately. The open edge structures can easily be separated from closed edge structures (polygons) by applying an 8-connected shrinking algorithm. The open edge structures shrink to isolated points and closed ones are not affected. By eliminating isolated points after shrinking, the array will have only closed polygons. To obtain only open edge structures, the array containing closed polygons is subtracted from the array of all edge structures. The sequence numbers obtained by this algorithm have been coded as gray values and shown in Fig. 7 (a-b). The sequence information, label, length, and direction are retained only at the line-seed locations for further processing.

Matching of edge structures:

The four edge structure attributes, namely, label of each edge structure, sequence numbers of line segments, direction value of each line segment, and length of each line segment, are stored in arrays LBL, SEQ, DIR, LEN. In these arrays, all locations except the line-seed points (defined above) contain 0's. Since this data is quite sparse the matching can be significantly improved by packing the measurements in the adjacent Processor Elements (PE) of the MPP. This is accomplished by sorting the edge structure attribute quadruplets using LBL, and SEQ as primary and secondary keys, respectively. A parallel bitonic sort algorithm [12] is used for this purpose. The sorting brings each edge structure attributes into the adjacent PE locations. The quadruplet (LBL, SEQ, DIR, LEN) completely characterizes the edge structures. Since DIR, LEN are sorted using LBL and SEQ as primary and secondary keys, respectively, the order in which they occur in adjacent PE's is a complete description of the edge structures. The matching algorithm essentially looks for similar list of attributes, DIR and LEN. The array, DIR contains the directions of line segments. The angle between the adjacent sides is used for matching. This can be obtained by a single absolute difference operation of DIR values contained in adjacent PE's. If DIR alone is used, the polygon matching is not affected by rotation as well as scale changes. It is necessary to use both DIR and LEN to ensure that significantly different scales of similar objects are not considered identical.

The sorted arrays (LBL,SEQ,DIR,LEN) for each of the two images are treated as one dimensional vectors for subsequent matching. (The *snake-shift* feature on the MPP is extensively used for this purpose). To permit matching of open edge structures in one of the images with closed polygons in the other without being sensitive to the (arbitrary) starting segments in the polygons and to avoid sensitivity to reversal of the sequencing of segments in the edge structures, the quadruplets for each edge structure are duplicated in the forward and reverse direction respectively, for the first and second images. Thus for example, the directional attributes for an n-sided edge structure in the first image are stored as D1, D2, .., Dn, D1, D2, ..., Dn and for an m-sided edge structure in the second image as d1, d2, ..., dm, dm, dm-1, ..., d1. The parallel matching of polygons proceeds as follows.

1. Normalize DIR and LEN features of each polygon

2. Perform global correlation of DIR features and LEN features separately for both images. That is compute C_d = DIR1⊗DIR2 and C_l = LEN1⊗LEN2, where ⊗ denotes correlation. This is done efficiently using Fast Fourier Transform.

3. Locate local peaks in C_d and C_l.

4. Shift DIR2 by an amount equal to the peak coordinate and subtract from DIR1. Find sum of absolute differences within a local window of size 5 around each point.

5. Where the sum of absolute differences is less than a predetermined threshold, load the corresponding labels in

the output array.

6. Set DIR1 and DIR2 at locations of matched labels to 0.

7. Repeat steps 2 to 6 until no more matches are possible or DIR1 or DIR2 contains all 0's.

RESULTS AND DISCUSSIONS

Using the matching technique described in this paper we are able to match ice floes which have undergone significant translations and rotations during the time interval between two images (Fig. 8). The edge extraction significantly affects the results of the algorithm. We have demonstrated using test patterns that our algorithm is able to match polygons accurately which have undergone significant translations, rotations, fragmentation, and merging. These synthetic images are shown in Fig. 9(a-b). The corresponding edge structures are shown in Fig. 10. The algorithm has been tested with subimages of ice floe images and found to yield satisfactory results.

In the present work, we have established correspondence among objects contained in the images by matching the sides having same subtended angles and lengths. Using the information so derived, it is possible to establish correspondence among the pixels and thus compute optical flow.

The computation times are data dependent. For the 128 x 128 test image, where the edge detection step was not needed the algorithm took approximately 500 msec on the MPP. For ice floe images, 700 msec of the MPP time was required including the time needed for the edge focussing algorithm. However, it is to be noted that for larger images than 128 x 128, one would process all 128 x 128 segments to obtain the edge structures and pack the attribute quadruples and then perform the matching.

REFERENCES

1. Fily M., and Rothrock D. A., 'Sea Ice Tracking by nested correlations', IEEE Trans on Geoscience and Remote Sensing,' Vol. GE-25, pp. 570-580, Sept 1987.
2. Vesecky J. F., Smandani, R., Smith M.P., Daida, J. M., and Bracewell R. N., 'Observation of Sea-Ice Dynamics Using Synthetic Aperture Radar Images : Automated Analysis,' IEEE Trans on Geoscience and Remote Sensing, Vol. 26, pp. 38-48, Jan 1988.
3. Collins M. J., Emery W. J., ' A Computational Method for Estimating Sea Ice Motion in Sequential Seasat Synthetic Aperture Radar Imagery by Matched Filtering,' J. of Geophysical Research, Vol. 93, pp. 9241-9251, Aug, 1988.
4. Davis L. S., 'Shape Matching Using Relaxation Techniques' IEEE Trans. on Pattern Anal. and Mach. Intell. Vol. PAMI-1, pp. 60- 72, Jan 1979.
5. Aggarwal J. K., and Duda, R. O., 'Computer Analysis of Moving Polygonal Images,' IEEE Trans. on Computers, Vol. C-24, pp. 966-976, Oct 1975.
6. Clark C. S., Conti, D. K. Eckardt, W. O., McCulloh, T.A., Nevatia, R., and Tseng, D. Y.,'Matching of Natural Terrain Scenes,' Poc of the IEEE Conf on Pattern Recognition and Image Processing, pp. 217-222, 1980.
7. Manohar M., Ramapriyan H. K., ' Connected Component Labeling of Binary Images on Mesh Connected Massively Parallel Processor,', to Appear in Computer Vision Graphics and Image Processing', Jan 1989.
8. Marr D. C., Hildreth E., ' Theory of Edge Detection,' Proc R. Soc. London, B 207, pp. 187-217, 1980.
9. Canny J., 'Computational Approach to Edge Detection,' IEEE Trans on Patten Anal. Mach. Intell. Vol. PAMI-8, pp. 679-698, Nov 1986.
10. Tilton J., 'Image Segmentation by Parallel Region Growing with Application to Data Compression and Image Analysis, Proc of 2nd Symposium on the Frontiers of Massively Parallel Computation, Oct 10-12, 1988.
11. Bergholm F., 'Edge Focussing', IEEE Trans. on Pattern Anal. and Mach. Intell. Vol. PAMI-9, pp. 726-741, 1987.
12. Dorband J. E., 'Sort Computation', Proc of 2nd Symposium on the Frontiers of Massively Parallel Computation, Oct 10-12, 1988.

NOTE: All figures appear as Color Plates IV, V, and VI, on pp. 696–698 of these *Proceedings*.