US Army Corps of Engineers
Topographic Engineering Center

Vision-Based Navigation for Autonomous Ground Vehicles

Larry S. Davis
Daniel DeMenthon

Computer Vision Laboratory
Center for Automation Research
University of Maryland
College Park, MD 20742-3275

May 1992

Sponsored by:
Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22201-1714

Monitored by:
U.S. Army Corps of Engineers
Topographic Engineering Center
7701 Telegraph Road
Alexandria, Virginia 22315-3864

TEC

U.S. ARMY TOPOGRAPHIC ENGINEERING CENTER

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
May 1992

3. REPORT TYPE AND DATES COVERED
Final Report May 1988 - May 1992

4. TITLE AND SUBTITLE
Vision-Based Navigation for Autonomous Ground Vehicles

5. FUNDING NUMBERS
DACA76-88-C-0008

6. AUTHOR(S)
Larry S. Davis Daniel DeMenthon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Vision Laboratory/Center for Automation Research
University of Maryland
College Park, MD 20742-3275

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Advanced Research Projects Agency
3701 North Fairfax Drive, Arlington, VA 22201-1714

U.S. Army Topographic Engineering Center
7701 Telegraph Rd., Alexandria, VA 22315-3864

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
TEC-0028

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes research on ground navigation, dynamic visual surveillance, parallel vision, parallel search, and parallel matrix operations. Specific problems addressed include planning safe paths on terrain, estimating range shadows, road reconstruction, dynamic object pose estimation and tracking; quadtree and pyramid algorithms, border following, Hough transformation, and graph matching.

14. SUBJECT TERMS
Border following, graph matching, Hough transforms, matrix operations, parallel search, path planning, pose estimation, pyramids, quadtrees, range imagery, road following, terrain navigation

15. NUMBER OF PAGES
83

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

NSN 7540-01-280-5500 Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std. Z39-18

Contents

List of Figures

PREFACE

This research is sponsored by the Advanced Research Projects Agency (ARPA), Arlington, Virginia 22201-1714 and monitored by the U.S. Army Topographic Engineering Center (TEC), Alexandria, Virginia 22315-3864, under Contract DACA76-88-C-0008, by the Center for Automation Research, University of Maryland, College Park, Maryland 20742-3275. The Contracting Officer's Representative at TEC is Ms. Lauretta Williams. The ARPA point of contact is Dr. Oscar Firschein.

1. Introduction

The research conducted under this contract focused on five main areas:

1. Applications in ground navigation. As a contributor to the ALV program during 1985–89 we developed a navigation system that was able to navigate a laboratory robot over a terrain board containing a simple road network. These algorithms were also brought to Martin Marietta and used to navigate the ALV over the Denver test track. In Section 2 we describe research on topics fundamental to cross country navigation and road following. We first describe massively parallel algorithms for route planning in digital terrain maps. What distinguishes this from previous route planning research in terrain is the incorporation of models for the presence of adversaries in the environment, and route planning criteria reflecting the need to move without being seen by these targets, yet being able to monitor their positions from time to time during navigation. The second part of Section 2 describes our research on the problem of filling in range shadows. We discuss why classical interpolation methods are not appropriate for this problem, and present new methods for filling in the shadows. The third part of Section 2 describes our research on 3D reconstructions of roads from images.

2. Core research on autonomous visual navigation. Section 3 of the report describes the research we performed on a project called RAMBO (an acronym for Robot Acting on Moving BOdies), on which we developed Connection Machine algorithms for low-level vision, intermediate level vision and visual planning to allow a mobile robot to pursue (in simulation) a moving three dimensional target through space in order to maintain visual contact with points on the surface of the target. The RAMBO project has led us to consider many fundamental problems in mapping vision and planning algorithms onto massively parallel machines, as well as develop new vision algorithms for basic problems such as pose estimation, motion estimation and motion planning.

3. Parallel vision algorithms. If autonomous systems are to operate in real time, then their vision problems must be solved using parallel algorithms. We have focused on the effective utilization of the Connection Machine for solving vision problems at all levels of analysis—low level vision (preprocessing, local feature extraction), intermediate level processing (matching, stereo, motion) and high level processing (recognition and planning). In Section 4 we describe our research on parallel vision algorithms. This research includes general multiresolution image processing methods for the Connection Machine, as well as quadtree algorithms.

4. Search plays a central role in almost all artificial intelligence systems, and especially at higher levels of analysis in vision systems. We have developed a new parallel search algorithm for MIMD machines. Experiments with the algorithm have resulted in linear speedups up to large numbers of processors (96) on realistic scheduling problems. The algorithm and some representative experimental results are presented in Section 5.

5. Parallel matrix operations. Many complex image processing operations, including restoration and reconstruction from projections, are based on common matrix opera-

tions (multiplication, inversion, etc.) We have developed several Connection Machine matrix manipulation algorithms. They are described in Section 6.

While the research described in Sections 2 to 6 forms the greater part of the work conducted under support of the contract, there were many other projects whose results were reported under this contract during the past four years. Some of these were the Ph.D. dissertations of students, and others were research projects conducted by visitors to the Laboratory that we felt best fit into our research in navigation. In Section 8 we present brief descriptions of these research projects.

2. Ground Navigation

While the dominating problem in visual navigation during the 1980's was road and road network following, the 1990's are seeing a greater emphasis on cross country navigation. In this research program we have developed new algorithms for vision and planning specifically designed to support applications in cross country navigation. Here we describe two such projects. The first, initiated in mid 1990, involves developing massively parallel algorithms for planning routes for vehicles in natural terrain. Our research differs from most previous research in a variety of ways. One of the most important is that we form plans with respect to models of adversaries moving in the environment. We can also plan routes for collections of vehicles that must operate under specific transport protocols.

We also describe, more briefly, in this section, research on new interpolation methods for filling in range shadows for range images taken in natural terrain. This research is relevant to local path planning in rough terrain where one would want to make the most informed guess about hidden parts of the environment.

2.1. Planning a Safe Path on Digital Terrain

2.1.1. Introduction

We now describe our work on planning a safe path in the presence of several moving adversary agents. The algorithms were implemented on the Connection Machine, using a digital terrain map. We first define the problem, then briefly present the algorithm.

2.1.2. Problem Definition

Given a digital terrain map and information about a friendly agent and adversary agents moving on the ground, we want to find a path for the friendly agent to a final goal that is hidden from the adversary but close to points of high visibility, taking advantage of the terrain.

The information available for the friendly agent includes:

- its current position;
- location of the final goal;
- maximal speed of motion;

The information about each adversary agent includes:

- its current position;
- its predicted motion trajectory;

Although we are able to plan a complete path at once, the feasibility of such a path is questionable due to the uncertainty of the prediction about the motion of the adversaries. Thus our approach is to set up a time interval over which we are fairly sure of the motion of the adversary, and make a subplan for that interval to a subgoal. The subgoal is chosen to be closer to the final goal and to have good observability in its vicinity, from which we can observe the activity of the adversaries to update our information about their motion. Then we iterate through this subplan process all the way to the final goal. There are two important time factors for each subplan:

1. Subplan interval: the interval of time we are planning for in each subplan;
2. Thinking time: the time required to generate a subplan. In our approach, this is a linear function of the product of the number of adversary agents and the length of the time interval we are planning for. We assume that the thinking time is known at the beginning of the subplan.

The criteria for a subgoal are:

1. It is safe at the end of the subplan interval (and preferably safe for a certain period);
2. It must be reachable at the end of the subplan interval through a safe path;
3. It is closer to the goal;
4. It has good observation points in its vicinity.

So the complete path planning contains the following loop:

```
WHILE final goal is not reached
 begin
      Pick a subgoal;
      Plan a path to the subgoal;
      Execute the subplan and update information;
 end.
```

The first two steps in the loop, namely the subplan process, are detailed in the following sections.

2.1.3. Algorithms

For each subplan, as a first thought, we may want to analyze the situation at the end of the subplan interval, pick a subgoal, and plan a safe path by computing the visibility map at each discrete time step and search through it. However, there are several serious drawbacks to this naive approach: when we pick up a subgoal, there is no guarantee that there is a path reaching that point at the end of the interval, even if the subgoal is within the mobility constraint. Furthermore, even if such a path exists, we may spend a lot of time trying to find it or fail to find it.

Considering these factors, we propose a different approach by taking advantage of the power of parallelism. Instead of first analyzing the situation at the end of the interval, we make an incremental computation of the safely-reachable region for each time step, denoted as RS_t, where t is the time index. Initially, t is set to the end of the thinking time, because that is the time at which the friendly agent starts moving. $RS_{\mathbf{end\_of\_think\_time}}$ is set to its current location. Then for each time step, we compute RS_t by first expanding RS_{t-1} according to its maximal speed, and then curtailing it by the regions visible from the predicted adversary positions at time t. After iterating this process, t is incremented to the end of the subplan interval, and it is guaranteed that there is a safe path to any point in the region $RS_{\mathbf{end\_of\_interval}}$. Then we evaluate each point in this region by the criteria mentioned in the previous section and choose the best point as the subgoal.

With this method for choosing a subgoal, the existence of a safe path to the subgoal is guaranteed, but how can we find this path, especially without an elaborate search? This problem can be thought of as a 3D path planning problem, with the RS's defining a corridor along the time dimension. We want to find a path through this corridor that satisfies the speed constraint of the agent, which is interpreted as the angle between its moving direction and the time axis in the 3D model. The solution is to build the path backward in time. We claim that for any point, say P, in RS_t, there exists a point Q in RS_{t-1} such that the agent is able to move from Q to P in one time step. This property comes from the way we build up the RS structure.

Thus we start at the subgoal and dilate it according to the speed constraint. The result defines the set of points the agent has to reach at one time step before the end of the subplan interval if we want it to be at the subgoal at the end of the interval. Then we determine the intersection of the set and the RS at that time. The result is a set of points that may be on our path. The property mentioned above guarantees that this set is non-empty. We have to pick one of them according to some goodness measure, and we pick the point with the least distance to the straight line between the last path point and the present location of the friendly agent, which tends to minimize the path length. Then from the chosen path point we repeat the dilation-intersection-selection procedure, until the time index is decremented back to the end of the thinking time. Since the current position is the only point in the first RS region, the path must end up at the current location.

The algorithm for the subplan process can be summarized as:

```
Algorithm Subplan
Begin
    { Step 1: finding the subgoal by forward dilation }
    $RS_{\text{end\_of\_think\_time}}$ = the current position of the friendly agent;
    For $t$ from the end of the thinking time to the end of the subplan interval
    begin
        $RS_t = dilate(RS_{t-1})$;
        Predict the adversary positions at this moment;
        Analyze the visibility from the adversary agents to the vicinity of the
            friendly agent;
        $RS_t = RS_t \bigcap Non\_visible\_region$;
    end ;

    Evaluate points in $RS_{\text{end.of\_interval}}$ and pick the best one as the subgoal;

    { Step 2: finding the path by backward dilation }
    $PATH_{\text{end\_of\_interval}}$ = the position of the subgoal;
    For $t$ from the end of the subplan interval back to the end of the thinking time
    begin
        $PATH_t = dilate(PATH_{t+1})$;
        $PATH_t = PATH_t \bigcap RS_t$;
        $PATH_t = best\_point(PATH_t)$;
    end ;
    return( $PATH$ );
End
```

2.1.4. Extensions

We have also investigated path planning issues such as the traversability of the terrain and the requirements for the friendly agent to observe the adversary at several points along the path and to remain close to other friendly agents, and to take risks on limited portions of the path in the absence of safe alternatives.

2.2. Range Shadows

In range images, there are regions behind obstructions that remain occluded (Figure 1). The sizes of these shadows depend on the relative elevation of the range scanner and the height of the obstruction. Many techniques have been developed for surface reconstruction. Most are based on weighted local averaging. There are more elaborate techniques that also incorporate surface slope in local height averaging. These techniques often yield an obviously *incorrect* result, in the sense that the elevation of the reconstructed surface is overestimated and it could not have been occluded in the first place. We propose an algorithm that does not yield such incorrect results, while it preserves the statistical properties of the surface. The method is best suited for random stationary surfaces, such as rough terrain.

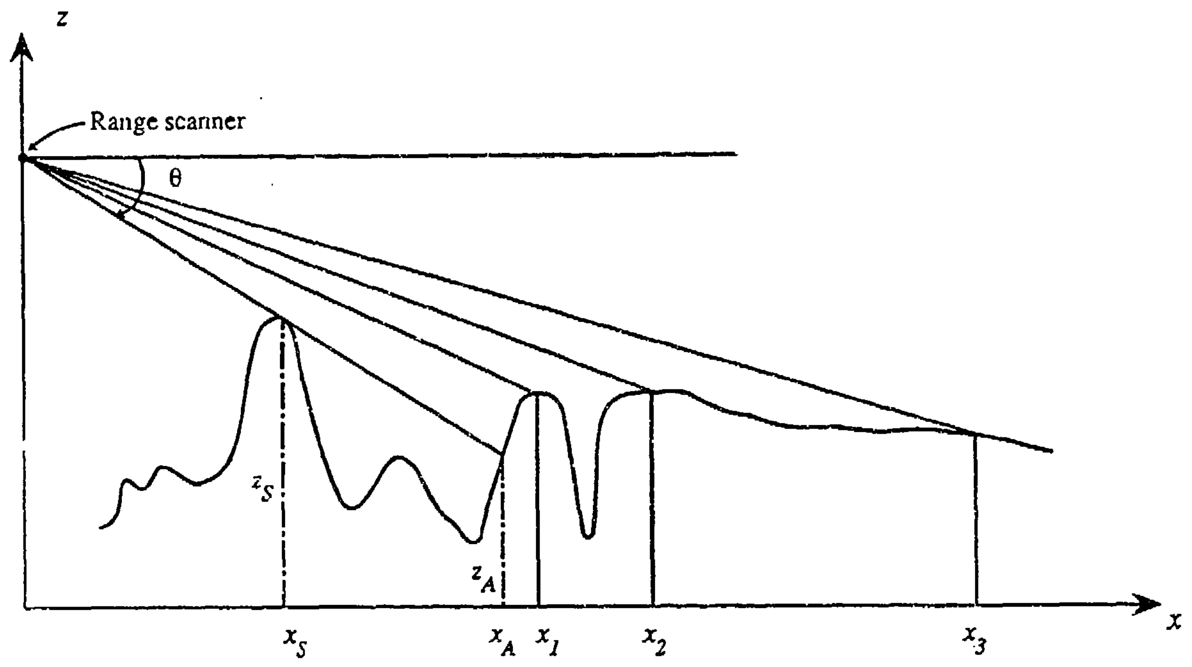

Figure 1: Schematic representation of a 1D surface and its range image shadows.

The algorithm estimates the surface elevation, z, at a given point $\vec{r} = (x, y)$. Regarding the elevation of a point in the range image shadow we have only one solid piece of information, namely, that the surface is below the shadow line, i.e., $z \leq z_S - |\vec{r} - \vec{r}_S|\tan\theta$, where θ is the angle of the shadow line with the x-axis, S is the point casting the shadow—how far below, we cannot know. In the absence of any other information about the depth of the shadow, it is reasonable to expect that the amount by which the surface lies below the shadow line is related to the variance of the surface elevation in the neighborhood. Therefore, as the simplest hypothesis, we take the point to lie *one standard deviation* below the shadow line, that is,

$$z = z_S - |\vec{r} - \vec{r}_S|\tan\theta - \sigma, \tag{1}$$

where σ is the standard deviation obtained from the variance

$$\sigma^2 = \sum_{i=1}^{n} w_i(z_i - \bar{z})^2 \tag{2}$$

In this equation, $\bar{z}$ and w_i are defined by the following expressions

$$\bar{z} = \sum_{i=1}^{n} w_i z_i, \tag{3}$$

where n is the number of nearest neighbors, z_i is the elevation of the i^{th} neighbor, and the weight w_i is inversely proportional to some power of the (Euclidean) distance $|\vec{r} - \vec{r}_i|$ of the point from its i^{th} neighbor

$$w_i = \frac{1}{N|\vec{r} - \vec{r}_i|^{\nu}}, \quad \text{with} \quad N = \sum_{i=1}^{n} \frac{1}{|\vec{r} - \vec{r}_i|^{\nu}}. \tag{4}$$

N is the normalization factor, and ν is a positive integer which is usually taken to be equal to 2. The number of neighbors, n, is usually between 3 and 8, and is often set equal to 5.

The intuitive justification for this scheme, which we refer to as the lowering method, is that if the surface is smooth (small σ) then the chances are that the shadow will not be very deep. Obviously this scheme is well suited for reconstruction of random stationary surfaces, such as rough terrain, where the local elevation variance is a characteristic property of the surface and is meaningful. Nevertheless, even for reconstruction of surfaces behind obstacles on a flat road the results are remarkably good and definitely better than the weighted averaging technique (Figure 2).

2.3. Zero-Bank Algorithm with Global Optimization

We developed a new "zero-bank" method for the reconstruction of a road in three-dimensional space from a single image. The world road is modelled in a similar way as in our previous "zero-bank" method, i.e., as a space ribbon generated by a centerline spine and horizontal *cross-segments* of constant length (the *road width*) cutting the spine at their midpoint at a normal to the spine. Because reconstructions of cross-segments are now independent of each other, a global optimization of the road reconstruction can be used.

2.3.1. Background

Our previous zero-bank method was iterative, requiring the knowledge of a previous cross-segment of the road to compute a new cross-segment. The problem was that at each iteration step, up to three possible cross-segments were found by solving a cubic equation; each of these three segments could be used as starting elements for up to three new cross-segments at the next recursion step, leading to a tree of possible roads growing exponentially. To avoid exploring all these alternatives, we chose in our previous method to keep at each step only the most plausible cross-segment, namely the cross-segment giving the smallest changes of road slope and turn. These are local criteria, however, whereas the goal is to obtain the best *global* road reconstruction. The images of the road edges could be locally of poor quality, leading to the wrong branching choice at this place of the tree of possible reconstructions, and the rest of the construction downstream could suffer or be brought to a dead end.

2.3.2. The New Approach

In the new method, tangents to the road edges at the end points of cross-segments are assumed to be approximately parallel. Thanks to this reasonable addition to the road model, a recursive reconstruction is not required. Cross-segments can be found from any element of road image, independently of previous pieces of reconstruction. Several possible local solutions of cross-segments are still found, but in this case we do not need to make choices until all the available evidence from the image has been used. Then we can choose the best global reconstructed three-dimensional road passing through the alternative reconstructed cross-segments. This global optimization is not computationally expensive because we can apply a dynamic programming technique minimizing local slopes and turns.

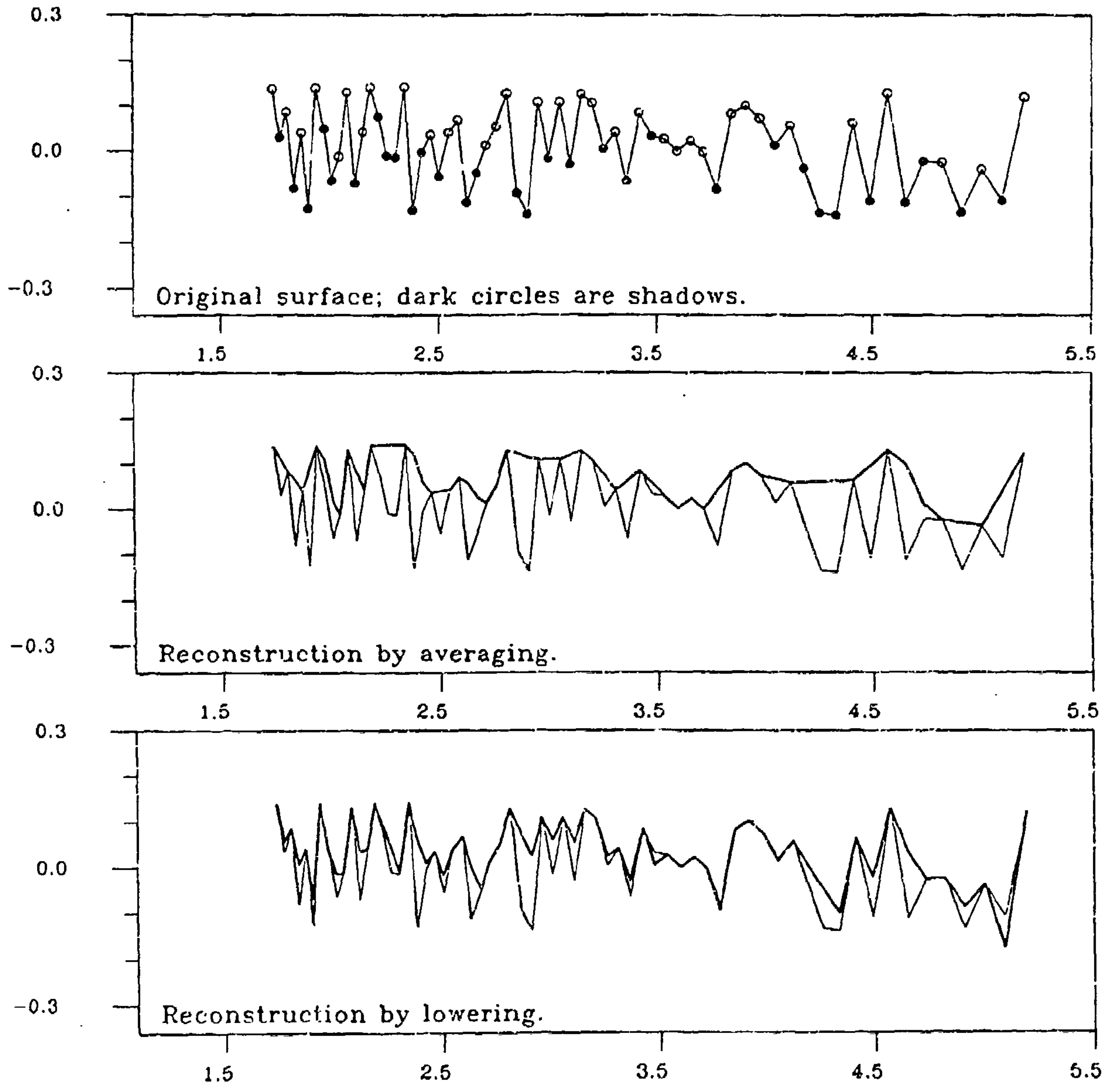

Figure 2: A digitized rough 1D surface and its reconstruction by the local averaging and lowering methods.

2.3.3. The Algorithm

A summary of the algorithm is now given; Figure 3 illustrates the road geometry reconstruction.

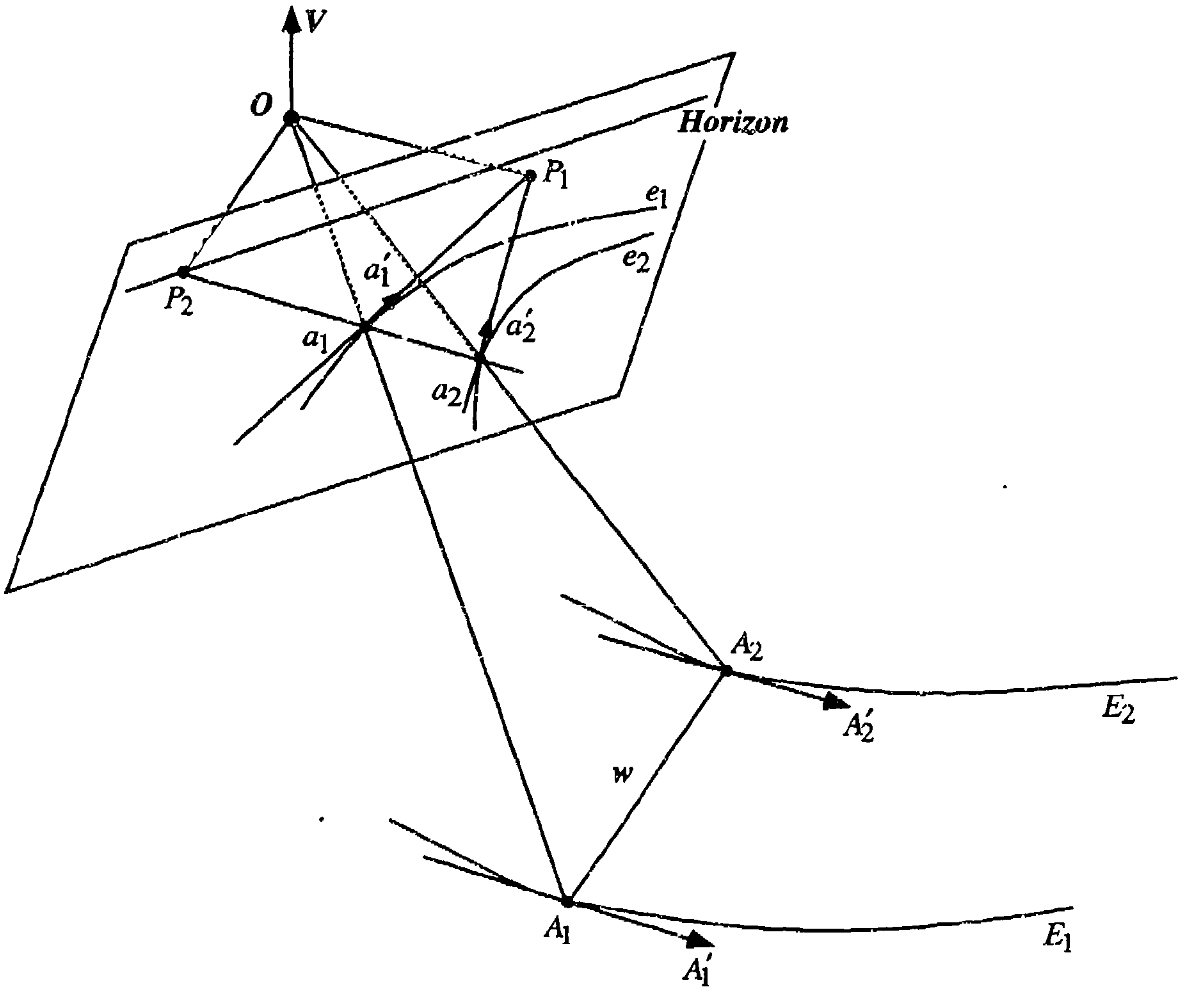

Figure 3: Reconstructing a world road from an image. The cross-segment of the world road is assumed horizontal and perpendicular to the tangents at its end points. The tangents are assumed parallel. An equation is found that must be satisfied by the matching points in the image and also involves the image tangents and the vertical direction.

1. In a preliminary step, not detailed here, some appropriate image processing techniques have isolated the two curves of the edges in the image, and a polygonal approximation has been found for each edge curve.

2. Picking image points anywhere on one image edge curve, we are able to find the points which are candidates for being *matching points* on the other image edge curve. (Two image points are called matching points if they are images of the end points of cross-segments.) We found an expression that two image points located on the facing image

edge curves and the tangents to the edge images must satisfy to be matching points. If a_1 and a_2 are matching points and $\overrightarrow{a'_1}$ and $\overrightarrow{a'_2}$ are the tangent directions to the image edges in these points, the following relation holds:

$$[\overrightarrow{V} \times (\overrightarrow{a_1} \times \overrightarrow{a_2})] \cdot [(\overrightarrow{a_1} \times \overrightarrow{a'_1}) \times (\overrightarrow{a_2} \times \overrightarrow{a'_2})] = 0 \qquad (5)$$

For edge curves approximated by polygonal lines, the matching point a_2 can be on a line segment, and its position between the end points of the line segment can be expressed by a number between 0 and 1, whereas its tangent vector $\overrightarrow{a'_2}$ is constant; or the matching point a_2 can be at an end point of a line segment, with a constant position but with a tangent angle which can be expressed by a number between 0 and 1 within the range of angles of the two adjacent line segments. For one point picked on one image edge, we check for each of the line segments of the other image edge if a matching point belongs to that line segment, i.e., if our expression gives a linear coordinate between 0 and 1 for this line segment. Then we look for matching points at the nodes of the polygonal line by checking if the expression gives a number between 0 and 1 for the tangent angle. We repeat these searches for several points a_1 picked on one image edge.

3. For each point picked on one edge image, the previous step can give several matching points on the other edge image. One of the reasons is that the images of the edges can be very rough and wiggly. Another reason is that the condition used is only a necessary condition for two points to be matching points in the image of the road we are seeing. This condition is local and it is up to us to pick up among the found matching points the pairs which are the most globally consistent, and discard the other pairs. The criteria of optimization are three-dimensional criteria; thus at this step of the algorithm, from the pairs of matching points, the corresponding three-dimensional cross-segments must be found. This correspondence is unique if the cross-segments are assumed horizontal and of known constant length. The constant length is the width of the road, and cannot be defined by this method. The assumed road width is a scaling factor in the reconstruction, whereas the optimization is based on angular considerations, which are independent of scaling. For driving a vehicle the road width must eventually be obtained from other methods, such as stored data about the road, the "Flat Earth" method, or close-range methods such as stereoscopy or time-of-flight ranging.

4. The group of matching point pairs corresponding to a single point chosen on one edge is the image of a group of world cross-segments obtained at the previous step, and the world road can go through *at most one* of these cross-segments. If a sequence of points along one road edge is taken, a sequence of groups of cross-segments is obtained, and the world road must go through at most one of the cross-segments of each group, in the same order as the sequence of points chosen on the first road image edge. Each cross-segment can be represented by a node of a graph. A path must be found in the graph, which visits each group in the proper sequence and goes through at most one node of each group, and which maximizes an evaluation function which characterizes

a "good road". The total evaluation function is the sum of the functions of each of the arcs of the graph. The evaluation function for an arc is the sum of weighted criteria, which grade the choices of individual cross-segments and the neighborhood of consecutive cross-segments, based on angular considerations. It would also seem useful to introduce constraints such as a requirement for small differences of slope between successive patches, but this type of relation involves three successive cross-segments and complicates the interaction graph. The previous unary and binary criteria actually appear to be sufficient for discarding unwanted nodes. Dynamic programming is appropriate for this type of path optimization, and, compared to a brute force search for the best path, greatly reduces the complexity of the search.

2.3.4. Experiments

Two types of experiments were performed: (1) reconstruction of roads from synthetic road images and comparison with the road models used to create the synthetic images; and (2) reconstruction of roads at Martin Marietta, Denver, Colorado, from video data obtained from the ALV, with comparisons of the results with the reconstructions obtained by data fusion between video data and ERIM scanner range data.

In the synthetic data experiments, a criterion of *navigability* of the reconstructed road was defined, as the percentage of actual visible road that a vehicle half the width of the road can follow without driving its wheels on the edge, if the vehicle were following the reconstructed road given by the algorithm. For all configurations of slopes and image noise tested, the navigability of the reconstructed road is significantly better with the new zero-bank method than with the previous method or the Flat-Earth approximation.

In the experiment with actual road images at Martin Marietta, Denver, Colorado, the "ground truth" was given by a method combining range data and video data, developed by Morgenthaler and Hennessy. The road edges are detected in the video image. Considering the line of sight of a road edge point, the world point of the edge is the point where this line of sight intersects the ground. The line of sight has a range image (*epipolar curve*) which can be calculated and superposed on the range image of the ground. The intersection of the line of sight with the ground corresponds to a point on the epipolar curve point with the same range as the ground point of the ground range image. From the ranges of several road edge points a three-dimensional profile of the road edges is obtained. This is illustrated in Figure 4.

Reconstructions were produced for around 50 road configurations including combinations of turns and slope changes. Both methods proved quite robust, giving plausible and consistent road reconstructions for all these tests; however, the ERIM laser ranger has a limited range of action. Only the first 15 meters of the road could be reconstructed by the fusion method. The reconstruction by the zero-bank algorithm extended at least twice as far in most road configurations. In the short stretch where both reconstructions were available, the agreement was considered good in top view (Figure 5). Differences of elevations appeared in side view, although the difference would probably not have resulted in different steering of the vehicle. The zero-bank algorithm used a flat earth approximation in the first segments of the road to remove the scale-range ambiguity inherent to this algorithm. The flat-earth plane is calculated as an extension of the plane of the points of contact of the vehicle wheels

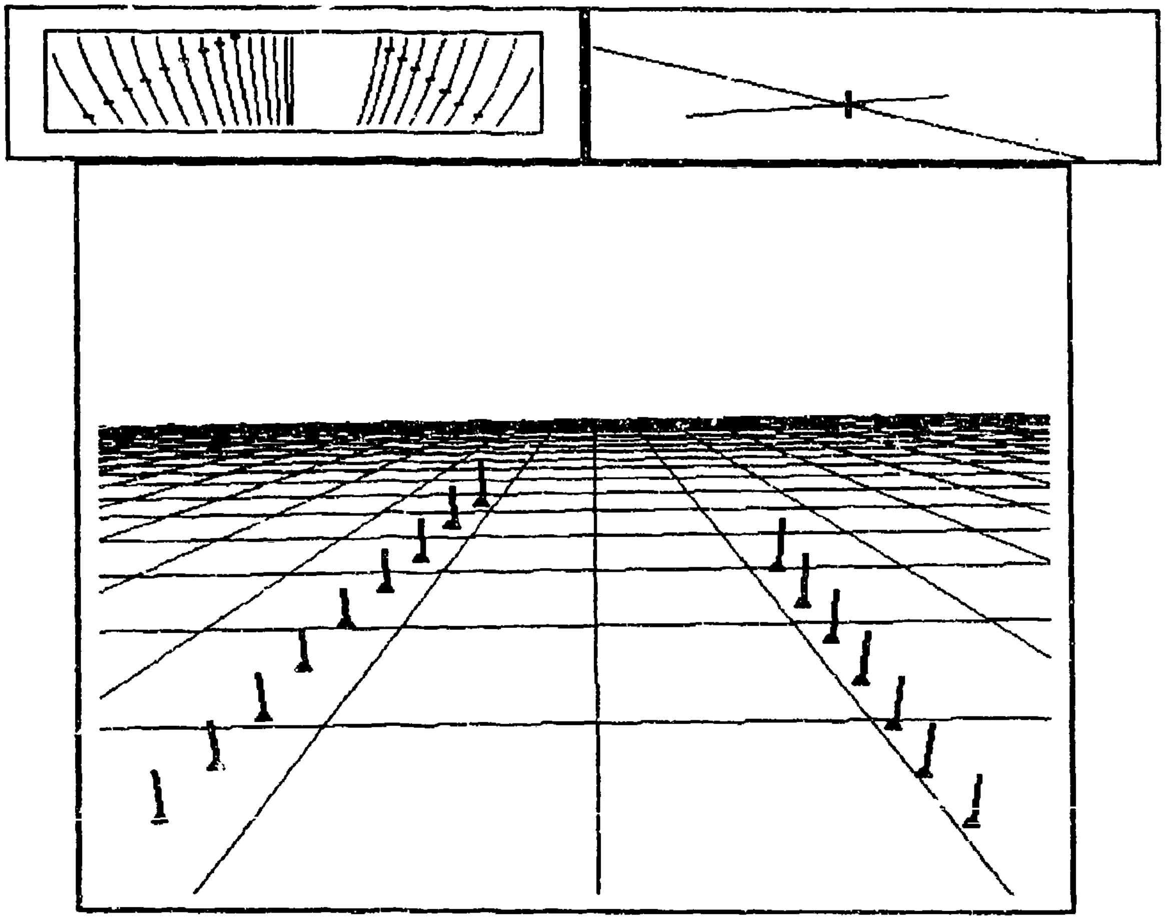

Figure 4: Results of range/video fusion showing (a) the epipolar arcs drawn within the ERIM image, (b) profiles showing the camera pixel ray and the elevation derived from the ERIM data along an epipolar arc, and (c) a perspective view of the resulting scene model.

with the ground. If the vehicle is on a local bump, the actual road will be lower than the result of this approximation. The laser ranger will of course detect this lower profile, and the fusion algorithm should produce a correct road profile.

2.3.5. Conclusions

Experiments with synthetic data show that the new zero-bank method gives useful information about road profiles even far away from the vehicle. Experiments with real data and comparisons with road reconstructions obtained by fusion between video data and range data show a good agreement in the short range in which range data are available, provided the scaling factor not defined by the zero-bank reconstruction is well chosen. Until range scanners covering a field as large as the field of view of video cameras are developed, the two methods can profitably be used together. The fusion algorithm can give the ground truth on the first 15 meters in front of the vehicle. The zero-bank algorithm can use this ground truth to remove its scale-range ambiguity, and extend the road reconstruction to most of the camera field of view.

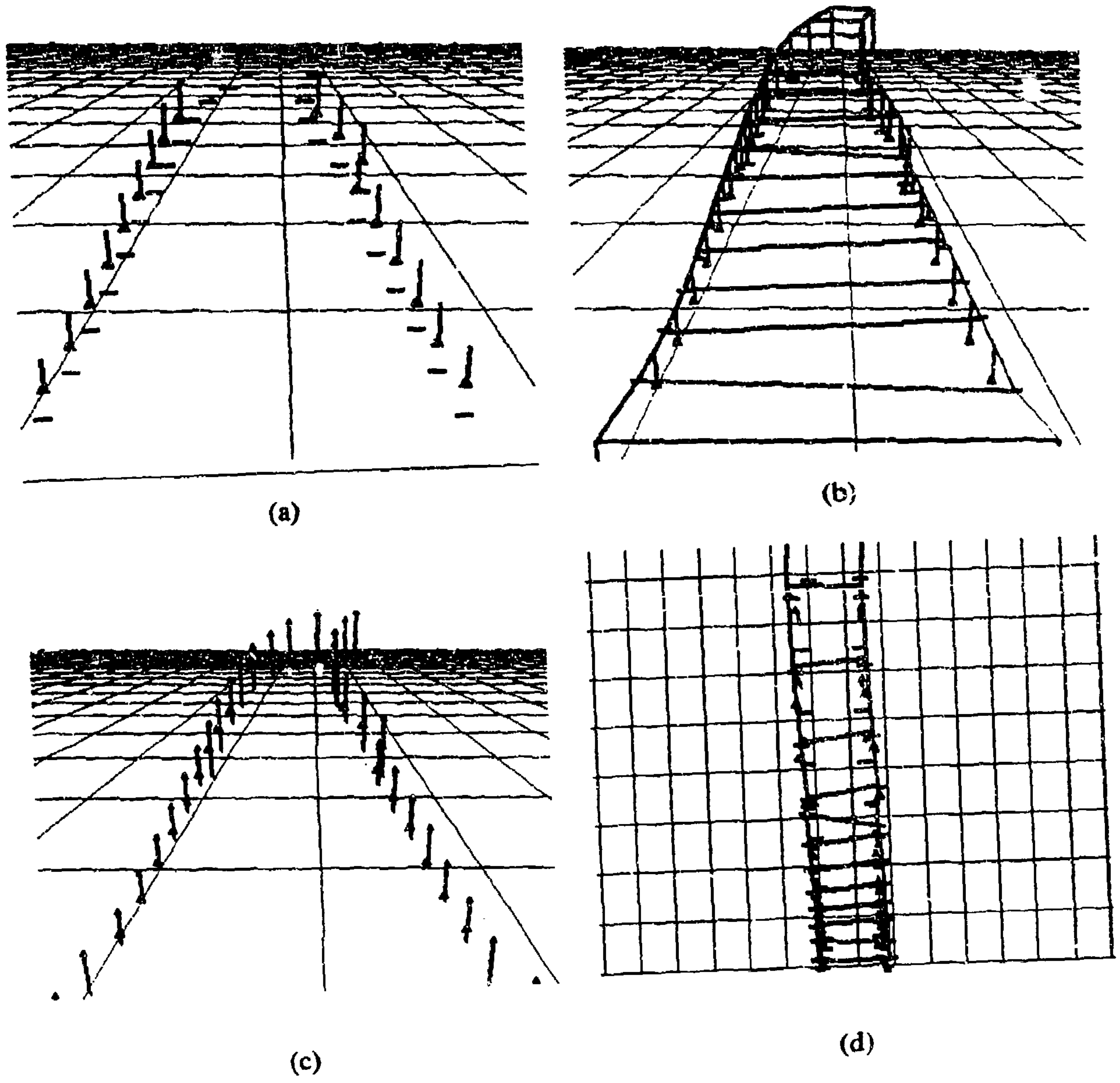

Figure 5: Road reconstruction results on a slightly curved, upsloping road scene. (a–c) show the fusion algorithm compared with: (a) the flat-earth algorithm, (b) the modified zero-bank algorithm, and (c) the hill-and-dale algorithm. An overhead view of all four approaches is shown in (d).

3. Dynamic Visual Surveillance

3.1. Overview

This section describes our research in the area of visual surveillance, in the context of the project we called RAMBO—Robot Acting on Moving BOdies. RAMBO is a framework designed to explore issues in high level vision and visual planning for an autonomous robot tasked to perform some visual surveillance activity on a moving object. We have developed algorithms for object pose estimation, object motion estimation, and visual planning. Many of these algorithms are massively parallel algorithms that have been implemented on a Connection Machine CM2. We describe an overall system architecture for the types of visual surveillance tasks that RAMBO is to perform, and then provide an overview of our research in vision and planning.

Specifically, we assume that the environment contains an object of known three dimensional structure. On the surface of this object are targets for visual surveillance. The object is in motion, and a general model of its motion is known. RAMBO's goal is to develop and modify a specific model for the motion of the target, and to use this motion model to construct a sequence of trajectories that will allow it to sight some set or sequence of the visual surveillance targets. We assume that the object will not change its motion in an effort to evade RAMBO. Computer simulations allow us to consider a wide variety of situations in which many parameters of the system (e.g., accuracy with which RAMBO can control its own motion, accuracy of various image analysis procedures, etc.) can be controlled and evaluated. They also allow us to bypass many difficult intrinsic and extrinsic calibration problems that would have to be solved in order to conduct meaningful quantitative experiments using real robots.

The vision algorithms developed as part of the RAMBO project are massively parallel algorithms designed to operate on the Connection Machine. In fact, another important goal of our research is to better understand how to integrate visual computations at many levels of analysis on massively parallel machines.

RAMBO's architecture has been heavily influenced by our experience with the Autonomous Land Vehicle project. As part of that project, we developed a system for visual navigation of a ground vehicle over roads and road networks. The organization of that system had the following important characteristics:

1. It operated in one of two control modes. Ordinarily, the system operated in the feed-forward mode of processing. Here, each control cycle included three stages of analysis:

   a) Prediction, in which the current road model (constructed on the basis of previously analyzed images) and an estimate of the vehicle's location in the model were used to generate predictions concerning the image locations of key road features (e.g., road boundaries and markings).

   b) Item verification, in which specialized image processing algorithms were employed to verify these predictions. These algorithms were applied to small search windows generated by the prediction stage.

c) Extension, where the road model was extended out towards the horizon using new information provided by the current image. If the verification stage failed, then the system switched to a bootstrap mode of processing, in which the vehicle would come to a standstill and perform a more global and time consuming analysis of its current image in an attempt to relocate itself relative to the road. This bootstrap mode of processing was also used to initialize the vehicle's road model.

2. Our system navigated on the basis of a three dimensional reconstruction of relevant parts of its environments. Its motions were planned and executed in a three dimensional coordinate system. This was, to a large extent, dictated by the relatively long time required to execute a single control loop of the system (this time was, of course, dominated by image processing), and a system with a substantially higher processing rate might have been able to perform road navigation solely on the basis of, say, visual directions to road boundaries.

RAMBO's organization is very similar to that of our road navigation system (see Figure 6). In a feedforward mode of processing, RAMBO first predicts, on the basis of its current motion model for the target and its own egomotion model, which target features (polyhedron vertices) should be visible in the image, and where they should be located. In fact, unlike the ALV system which acquired an image as soon as a control cycle was completed, RAMBO also needs to reason about when to acquire its next image, since it might not be possible to move along a trajectory from which a sufficient set of target features would always be visible. RAMBO also uses an estimate of errors in its motion models to choose appropriate size search windows for the verification process. These search windows are the byproduct of a Kalman filter employed to estimate target motion and extrapolate target pose.

In the verification phase of processing, we have to perform local searches of the prediction windows for the target features. If an insufficient number of such features were detected, then we would regard the analysis as having failed, and would move away from the target to a safe location from which we could attempt to reinitialize our model of its motion using a bootstrap-like processing. Ordinarily, the locations of the detected vertices are used by a Connection Machine pose estimation algorithm to estimate the location and orientation of the target in space.

The motion model for the target is then verified and extended by a Kalman filtering algorithm that takes the recent history of pose estimates and develops an updated model for the relative motion of the target. Based on this new model of target motion, RAMBO's current trajectory is modified (if necessary) by algorithms that find minimal trajectories to target locations. A next viewing position is then chosen based on potential visibility of target features and the estimates of errors in the motion models.

3.2. Implementation of Vision Algorithms

We briefly describe the vision algorithms developed for the RAMBO project. It should be noted that these algorithms have not been integrated with the trajectory control algorithms described below

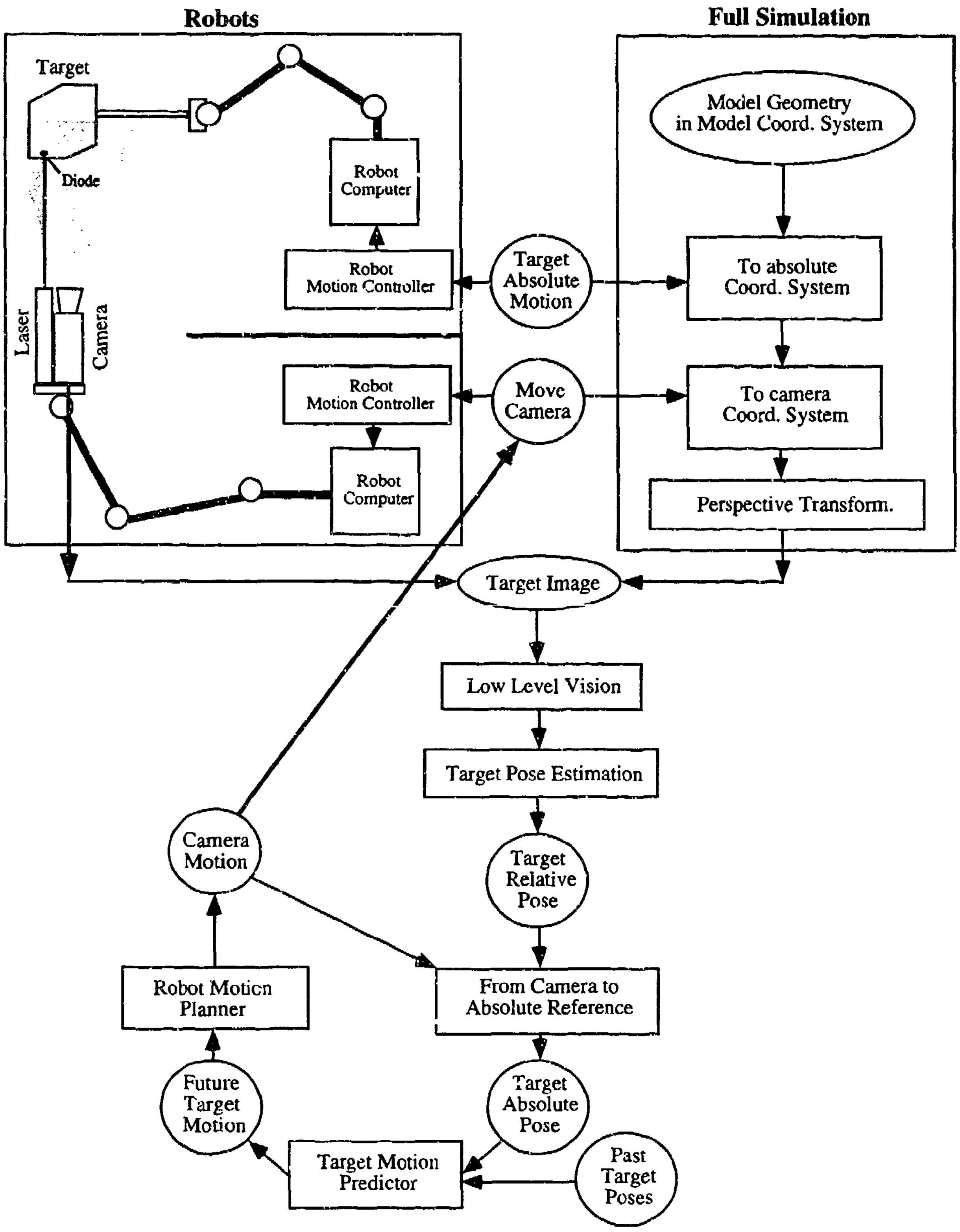

Figure 6: Vision-based control loop for a robot acting on a moving body.

3.2.1. Feature Detection

RAMBO's target is a simple, convex polyhedral object. The background is kept visually simple to simplify the low level processing. The image is first processed using a sequence of conventional image processing algorithms—a local edge detection operator is first applied, the magnitude of the edge detector is thresholded and the binary image is thinned so that all edges are part of simple chains or are vertices where such chains meet. RAMBO's pose and motion estimation algorithms are based on vertex matching, so the polyhedron vertices are detected in one of two ways. We have developed a simple corner detection algorithm that breaks the neighborhood of each edge pixel into a small number of sectors and identifies as corners those points that:

a) have edges in two or more noncollinear sectors, and

b) are local maxima with respect to a simple measure of curvature.

Alternatively, corners can be detected by a sequence of Hough transform, prediction of corners from pairs of clusters in the Hough transform, and verification through some additional image processing. The latter approach is more sensitive and accurate in its location of corners, but the former is much faster.

The feature points detected in an image are grouped into triples (the *image triangles*). Each image triangle can be described by one of its vertices (the *reference vertex*), the length of the two adjacent sides, and the angle between them (the *reference angle*). These adjacent sides do not necessarily have to correspond to actual edges in the image, and all distinct triples of points could be considered, with each triple of points producing three such image triangles. However, if the feature points are vertices, it is useful to only consider image triangles in which the adjacent edges of the reference vertex are actual edges, and to only match these image triangles to object triangles with similar characteristics. This increases the proportion of good matches over the total number of possible matches.

3.2.2. Initial Pose Calculations

The detected triangles are used to estimate the instantaneous six degree-of-freedom pose of the target. Our Connection Machine pose estimation algorithm is based on matching triangles of image corners against triples of target junctions. For perspective imaging, two possible pose estimates for an object can be recovered from a fourth degree equation determined by the correspondence between an image triangle and a triangle of the object. However, when RAMBO analyzes its first image, it does not have any *a priori* knowledge of the correct correspondence between the detected image triangles and the actual triangles of the object. In this case, the system uses all the possible combinations of target triangles and image triangles and clusters the pose estimates for these combinations.

Approximated Pose of a Known Triangle from its Image

Solving a fourth degree equation for each image triangle to object triangle correspondence, and picking the two correct triangles among the eight triangles given by the quartic solutions is time-consuming. Also, the fourth degree equation is sensitive to roundoff error for certain image parameters. For these reasons, we have developed an approximate perspective projection, *orthoperspective*, which provides simpler computations and reasonably accurate results. The idea is to project the figure onto a plane drawn through one of the vertices of the figure and perpendicular to the line of sight of this vertex (other approximations of this type have used planes parallel to the image plane instead). Then the image of the figure is approximated as the image of this projection (Figure 7).

Notice that if we rotate the camera around the center of projection to bring the optical axis to the line of sight of the vertex we considered, the image plane becomes parallel to the plane on which we performed the orthogonal projection. After this camera rotation orthoperspective is simply a scaled orthographic projection (also called *weak perspective*). Therefore, orthoperspective is equivalent to the following sequence of operations:

1. A virtual rotation of the camera around the center of projection to make a chosen line of sight coincide with the optical axis.
2. A scaled orthographic projection of the recentered image.
3. The inverse camera rotation to bring back the camera to its original position.

The camera rotation removes the dependence of the construction on the offset of the world object from the optical axis. For this reason orthoperspective is a better approximation to perspective than scaled orthographic projection for image elements which are not centered in the image plane.

With the orthoperspective approximation, finding the pose of a triangle from its image when the shape of the triangle is known reduces to solving a second degree equation.

Before the Connection Machine had its floating point coprocessors, we solved these equations beforehand and built lookup tables for each triangle of the model. The final implementation solves these equations as needed, on the assumption that routing communications to perform table lookups might be slower than direct computations. Comparisons are difficult because the new implementation is written in C-Paris whereas the lookup table version was in *Lisp.

Clustering of the Pose Parameters

Pose parameters for the object are obtained for each possible combination of image triangle and object triangle. The combinations which correspond to correct matches will have produced similar object pose parameters, and will thus be clustered in parameter space. Our clustering algorithm uses a virtual processor set with one processor for each data point in the pose space. It also uses a set of virtual processors to represent an orthogonal projection grid, and projections of the six dimensional pose space are computed using several axes. A parallel read-back scheme provides a voting mechanism by which high-density regions in a cloud of points can be identified. This new clustering method produces robust results, with few misidentifications of pose parameters under a variety of pose conditions.

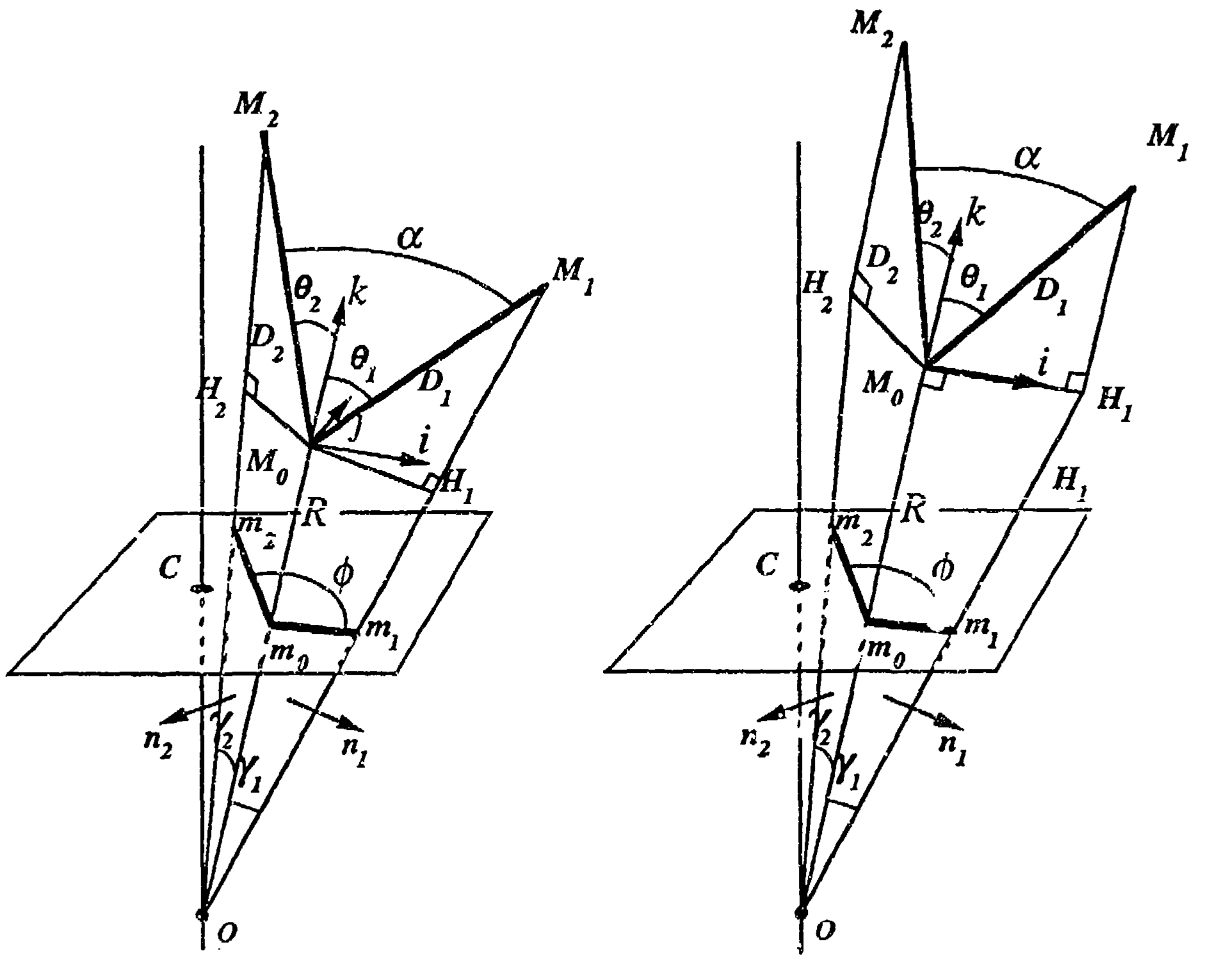

Figure 7: Exact perspective and orthoperspective for a triplet of points.

Verifying the Pose Estimate of the Object

This pose estimation algorithm cannot make much use of expectations of the target's pose that would be available during the feedforward mode of processing. We have therefore designed and implemented a Connection Machine hypothesize-and-test pose estimation algorithm:

Once we have generated expectations of target poses, we project visible model points with their local features onto the image where they are compared with the image points. The comparison is based on the locations of the image and projected points, on the number of adjacent edges and on their angles. If the projection of the target in the hypothesized pose passes these tests, its pose estimate is refined. We find the rotation R and translation T which minimize the sum of the squares of distances between the actual image features and the projections of the features of the object in its hypothesized position. We reach a minimum by applying Newton's method, starting from the estimated R_0 and T_0.

But even if all the correspondences between model and image points are correct, the optimized pose can be inaccurate, due to noise in the image. Theoretically, we could calculate the exact probability distributions of the pose parameters assuming, for example, bivariate normal location error. This would require, however, the solution of fourth degree equations, with complex analytic functions as a result.

We prefer to calculate an estimate of the reliability of the pose parameters by applying intermediary results of the pose estimate optimization. We perform one Newton iteration assuming bivariate normal noise added to each image of the model point. As a result we obtain normal distributions that estimate the true probability distributions of the pose parameters.

3.2.3. Feedforward Tracking and Pose Calculation by Kalman Filtering

The object pose could be estimated independently for every image frame in the sequence by the techniques presented in the previous paragraphs, and used to compute the motion. However, pose estimation starting from raw image data is a computationally expensive procedure that one would prefer to avoid. This is possible once satisfactory initial estimates of the motion parameters have been been obtained as described above. Then the future poses and the image locations of the features that correspond to known points of the model can be predicted, greatly reducing the computational cost. Based on the known correspondences, straightforward pose and motion determination methods can be used.

The motion parameters may change with time and the measurements obtained from the images are not infinitely accurate due to discretization. Knowledge of these uncertainties must be incorporated into the estimates in order to judge the chances for successful task execution.

Kalman filtering has been shown to be a powerful tool in similar problems, e.g., in tracking objects in monocular image sequences and in robot navigation. In our case the camera is in the hand of a moving robot and an accurate geometrical model of the target object is available, while its motion model contains uncertainties.

Uncertainty Regions of Feature Points

When tracking the object feature points in the image plane, it is useful to limit the search to regions where they are most likely to occur. By assuming the measurement function $F(*)$ linear, the uncertainty windows of predicted feature points can be approximated using the covariance matrix $V_{k|k-1}$ modeling the prediction of the measurement uncertainty

$$V_{k|k-1} = J_F(u_{k|k-1})Q_{k|k-1}J_F^T(u_{k|k-1})$$

where $Q_{k|k-1}$ is the estimation covariance matrix of the Kalman filter and $J_F(u_{k|k-1})$ is the Jacobian of the measurement function $F(u_{k|k-1})$, or the matrix of first order derivatives of feature point coordinates in the camera frame with respect to the state variables in the motion estimation frame. We have simply used the diagonal elements of this matrix to determine rectangular regions of interest.

The uncertainty windows also offer a simple confusion avoidance mechanism for feature points located close to each other in the image: points whose uncertainty regions intersect should not be used.

Motion Model

The object tracking performance of a robot depends critically on the match between an *a priori* model of motion and the actual object behavior, unless the system control delay can be made very short. Shorter control delays decrease the relative contributions of unmodeled changes of motion parameters per time unit. This results in smaller Kalman gains, smoothing the variations due to measurement noise. However, image acquisition and mechanical delays determine the minimum control delay of the system that cannot be shortened by computational solutions. Therefore, we need to consider the motion modeling problem in more detail.

When the camera sensor is mounted in the hand of a robot manipulator, the key questions in forming the object motion model are:

1. Should the state vector be represented in a moving coordinate frame or a global stationary world coordinate system (relative or absolute object motion)?

2. What motion parameters should be included in the state vector?

Coordinate System

If the target object is not able to perform significant accelerations, its motion in a fixed world coordinate frame can be modeled by constant rotation and translation. The possible small accelerations can be taken into account as noise. However, with the camera in the hand of a robot the motion model may not be independent of the mechanical solution. The measurements are done in a camera coordinate system and the transformation to the world frame, or vice versa, requires knowing the robot joint angles at the time of image capture. But the controllers of many off-the-shelf robot manipulators do not support obtaining the angles of different joints simultaneously during motion. This may significantly increase the uncertainties.

A practical compromise is to model the motion in a relatively slowly moving coordinate frame, in which the angle and translation parameters of the camera are simple to determine. For example, the origin of the coordinate system could coincide with the wrist joint of the robot.

Motion Parameters

If the state vector is represented in a moving coordinate system, modeling the expected relative accelerations as system noise may result in impractically high uncertainties. This also increases the sensitivity to measurement noise.

The uncertainties can be reduced by including the accelerations in the state vector and modeling the possible small changes of acceleration as noise. Unfortunately, a longer state vector increases the computational cost and relative motion uncertainty during the times when accelerations are near zero.

We have investigated two simple compromises that limit the drawbacks of higher dimensionality or higher noise level to the appropriate periods:

1. Variable-dimension state vector: If the estimation errors grow rapidly, the acceleration components are added to the state vector. The switch back to the constant velocity model is performed when the acceleration approaches zero or the errors go below a given threshold.

2. Variable system noise: A constant-velocity motion model is used, but when the estimation errors grow the system noise model components attributed to motion are inflated. When the errors decrease the noise component is returned back to the normal quiescent level.

The exact ranking of these solutions requires knowing the length of the control loop, including the image processing time and robot delays.

Mapping these techniques onto the system equation is straightforward. For simplicity, we have assumed that each motion component is independent and the trajectory of the object in the given coordinate system is straight.

Measurement Uncertainties

Our principle in dealing with the measurement noise has been to approximate the limits of certainty by the standard deviations of Gaussian distributions. Then straightforward methods can be used for transforming and combining the uncertainties. The feature point locations on the image plane are not arbitrarily accurate measurements because of discretization. As a result, the feature points in the image may deviate from their exact positions and the same point locations in an image may correspond to a range of poses.

We have considered the $2n$ feature point coordinate values as independent Gaussian distributed random vectors and denote their composite $2n$-by-$2n$ covariance matrix by V_v.

By assuming that the standard deviation of the feature point location coordinate error is 1 pixel, V_v becomes an identity matrix.

Note that there is no physical reason why the actual locations of the feature points would cluster around the centers of pixels. However, simulations with uniformly and Gaussian distributed noise showed hardly any difference in the resulting pose and motion estimates.

Pose Estimation Uncertainty

We start from known correspondences, so the pose parameter vector of a particular object is solved from the image feature point locations by the inverse of the measurement function

$$u = F_k^{-1}(v) = G_k(v)$$

If the pose determination function $G_k(*)$ was linear, the distribution of pose parameters, $V_u(u)$, would be Gaussian and could be obtained by

$$V_u(u) = J_G(v) V_v J_G(v)^T$$

where $J_G(v)$ is the Jacobian of $G_k(*)$ or its matrix of first partial derivatives. In the actual non-linear case this provides a local approximation of the pose distribution, hence the arguments u and v.

In this equation, the Jacobian J_G is obtained from the Jacobian of the measurement function:

$$G'(G^{-1}(x)) = \frac{1}{(G^{-1})'(x)}$$

or equivalently

$$J_G(v) = J_F^{+}(u)$$

where matrix pseudoinversion (+) is used, because J_F is not a square matrix except when $n = 3$.

Selection of Correspondence Points

The important problem of selecting the feature points used for estimation is now considered. Usually several feature points are available from an image frame, but because of various real-world constraints using only some of them is justified. With the test object that was used *random selection* of the feature points gave good results, and when the actual computational costs of the methods are taken into account, this approach becomes very attractive. Random selection does not favor any particular set of correspondences so this approach works with any number of feature points. The very low computational cost results in high speed and minimum motion modeling uncertainties, helping to offset the possible higher measurement uncertainty.

Note that the performance of the random selection technique depends on the distribution of the feature points on the object. In practice, it may be necessary to make the selection of certain important feature points, such as the ends of prominent extensions, more probable than that of the others.

The method of random feature selection was experimentally compared with three other methods. The first method, the minimum uncertainty method, attempts to choose the features which minimize the covariance matrix. It seems impractical for real time applications because of its computational cost, but can be used as a comparison reference. The second method, the condition method, chooses features that produce well conditioned measurement equations that minimize the sensitivity of the relative error in the system state estimate to the variations in the measurements. The third method, the pose information method, selects features which give the most accurate, or most informative object pose.

Random selection performs surprisingly well against these other methods with our test object. When compared to the minimum uncertainty method, the penalty is almost negligible with the selection of a *single* feature point. With only two points the method outranks both the condition of Jacobian and pose information approaches. The results become worse when more points are selected, however the low computational cost still acts as a significant compensating factor.

Higher sampling rates reduce the motion uncertainties and can in part be achieved by reducing the number of feature points used. Random selection is a promising method for this purpose. In practice, it is necessary to find a trade-off between better measurements and weaker motion model. In our experiments, maximization of the sampling rate has been the most rewarding direction of development.

3.2.4. Trajectory Control

One of the subtasks assigned to RAMBO is reaching a goal point located on the periphery of the moving target object. The autonomous system cannot just avoid the moving object as in classical obstacle avoidance problems. It must go to the vicinity of the object to reach the goal point without colliding with parts of the object.

What makes the problem difficult is that *the motion of object is not precisely known.* It is predicted by the Kalman filtering technique described in the previous section. Only the shape of the target object, and bounds on its speed and normal/tangential acceleration, are precisely known to the autonomous system.

Navigation in this context is understood as finding the successive positions that carry the robot toward the goal with an acceptable collision risk level. The method we developed requires calculating the probability of collision and using it as a safety measure to produce an optimal trajectory.

A solution to the trajectory control problem is then defined to be the solution of the following optimization problem:

> At present time t_n, based on the position D_n of the robot and the estimated target motion state, we wish to find the optimal *reachable* destination for the next time instant D_{n+1}. The optimality criteria are collision safety and proximity to the intermediate goal position.

The destination D_{n+1} is *reachable* if the changes in speed and direction required to get to D_{n+1} are within the robot predefined dynamic specifications. The intermediate goal is the estimated position of the goal at the next time instant if it is reachable, or the reachable position of the robot that would bring it to the goal point in the smallest number of time steps. We find that the proximity of a destination to the intermediate goal can be measured probabilistically and can be integrated into a cost function.

The probability of collision of the robot with the target object when moving from D_n to D_{n+1} is

$$PC\left(D_n, D_{n+1}\right) = P\left(D^r_{n+1} \in Sh(D^r_n)\right)$$

where $Sh(D)$ characterizes the *shadow* of the target with respect to a point D, i.e., the domain of space that an observer in D cannot see because it is hidden by the target (and therefore cannot reach by a straight line displacement from D without hitting the target). D^r_n and D^r_{n+1} are the positions D_n and D_{n+1} in a frame of reference of the target. In this frame of reference, obviously, the target position is completely defined, and the positions D^r_n and D^r_{n+1} are random vectors related to the target kinematic states S_n and S_{n+1} in the global frame. We have access to the marginal distributions of S_n and S_{n+1}. However, the kinematic state S_{n+1} is not independent of S_n, so that it is difficult to derive their joint distribution. Equivalently, it is difficult to derive a joint distribution $f_{D^r_n, D^r_{n+1}}(.,.)$ because the random vector D^r_{n+1} is not independent of D^r_n. The solution is to use the motion state evolution equation of the Kalman filter to express D^r_n and D^r_{n+1} as functions of independent random variables whose distributions are known. Their joint distributions are then expressed as the product of the marginal distributions of the independent random variables. We may then integrate to find the probability of collision.

Having defined a probabilistic measure of collision safety, we want to produce an optimal destination D_{n+1} for the robot at next time step t_{n+1}. One possible approach is to define the optimal destination as the one that minimizes the expected distance to the intermediate goal under an acceptable probability of collision. We sample the space reachable by the robot and find the position closest to the predicted goal position that can be reached with a tolerable risk of collision. Alternatively, the optimal destination can be taken to minimize a cost function that includes the probability of collision along with the probability of getting close enough to the intermediate goal.

3.3. Visual Planning

We have conceptualized RAMBO's planning activities as occurring at four hierarchical levels of analysis:

- Subtask sequencing—RAMBO is generally tasked to visually sight a set of locations on the surface of the target. If the ordering of subtasks is not specified in the problem description, then one must be chosen during task execution. While it would be possible to identify an *optimal* subtask ordering once some initial target motion model is established, we adopt instead a *greedy* approach since the future motion of the target is only weakly constrained by its current motion, and the cost of solving the combinatorial optimization problem is very high.
- Goal point identification—Once a subtask is chosen, RAMBO must determine a goal point (or small region in space) in the target's coordinate system from which it can

sight the target surface location. Generally, RAMBO would like to maximize its distance from the target while sighting, but a variety of factors must be considering in choosing a goal point. These include an estimate of the accuracy of the target motion parameters, RAMBO's accuracy in controlling its own motion, and the amount of real-time visual monitoring that RAMBO would need to perform in order to guarantee the safe execution of the plan.

- Generation of approach trajectory—Once a goal point is chosen RAMBO next computes a trajectory that will take it from its current trajectory to the goal point. This so-called reaching trajectory may either bring RAMBO into zero relative rigid body motion with respect to the target, or to a stationary position at the goal point. It should be chosen so that the target would always be in the field of view if the target motion model were correct.

- Monitoring and failure recovery—RAMBO must continually monitor the target to update its motion model. If the target's motion changes then RAMBO has to decide if a new reaching trajectory should be determined, if a new goal point should be chosen, or if the subtask ordering should be changed. It is also possible that the target may leave RAMBO's field of view (or RAMBO may fail to detect the target even though it is within its field of view). In this case RAMBO must initiate a visual search strategy to reacquire the target.

Our actual planning system is only able to deal with a small subset of these problems. RAMBO makes the simplifying assumption that a complex goal can be decomposed into a sequence of simple subgoals. All subgoal points required for each complex action on a target can be predetermined and stored in a database of actions specific to each target. Each goal point is defined by its pose in the object coordinate system.

As RAMBO progresses from one subgoal to another, it will generally have to change the trajectory along which it moves, so that at some time we would want RAMBO to launch from its current goal trajectory and to land at some subsequent time, $t_0 + T$, on a new goal trajectory. The duration T is referred to as the reaching duration. Once t_0 and T are chosen, then a reaching trajectory that takes RAMBO from its original goal trajectory to the new goal trajectory can be determined by using, for example, a parametric cubic spline that ensures continuity and smoothness at both takeoff from the original trajectory and landing on the new goal trajectory.

4. Parallel Vision

4.1. Overview

Massively parallel computers, such as the MPP, DAP, and Connection Machine, are ideally suited for low level vision operations such as convolution, image arithmetic and image morphology. Machines without the rich interconnection structure of the Connection Machine are at a distinct disadvantage in supporting intermediate level vision processing. There are two main reasons for this:

- In many intermediate level vision algorithms, data must be combined from image regions that are physically separated. So, even an operation as simple as the Hough transform for line detection is cumbersome on a mesh connected machine because the *votes* from points lying along any line must be collected, eventually, at a single processor in the machine.

- In other intermediate level vision algorithms, operations are performed not on the image array itself, but on more general data structures. So, for example, to apply our pose estimation algorithm using triangles (previous section), we develop a representation of the image that is structured as a set of triples of edge junctions detected in the image by a corner detector. The junctions forming any triple may correspond to image points that are widely separated in the image.

The additional interprocessor connections included in the Connection Machine's hypercube are critical to overcoming these problems. First, the diameter of the hypercube (i.e., the greatest number of wires one must traverse in moving information from one processor to another) is only the logarithm of the number of processors in the machine, as opposed to the square root of the number of processors in a mesh. This makes operations such as the Hough transform very efficient, and is also critical for efficient implementation of more general grouping operations.

Second, the hypercube connections support logarithmic implementations of fundamental parallel algorithms, such as grid permutations and scan operations. These operations form the basis for most of the grouping operations that RAMBO performs.

An interesting technical problem arises in focus-of-attention vision algorithms. This will generally involve determining relatively small image windows in which specific object features are predicted to appear, and then applying specialized image processing algorithms that are "tuned" to the expected photometric and geometric properties of the predicted features.

This can be accomplished in a straightforward way on the Connection Machine by simply selecting the processors associated with the window(s) to be processed, and applying the image processing operations to those selected processors. The remaining processors would be idle during this stage of computation. The disadvantage of this approach is that a large part of the Connection Machine would not be utilized. Similar problems occurred with earlier commercial machine vision systems, which required 1/30 second to apply a basic operation to a frame even if only a portion of the frame were of interest. Current systems have so-called *region-of-interest* operators, which essentially involve some additional addressing hardware that pipeline only a window of the full frame through the processing hardware. With such region-of-interest hardware, the time that it takes to apply a basic operation to a window is proportional to the number of pixels in the window, as opposed to being fixed.

We would like to achieve a similar economy of scale on cellular machines like the Connection Machine. There are two approaches that we can identify that could potentially lead to more effective utilization of the massive parallelism available in today's cellular computers:

- Partition the machine into clusters of processors, and assign each cluster to a single pixel in the window to be processed. This is the approach proposed in the so-called *fat pyramid* model for image processing on hypercube connected machines (next subsection). Our software Connection Machine implementations certainly indicate that any efficient instantiation of this concept would require extensive hardware support.

- Replicate the image window as many times as will fit into the available processor set. This is the approach that we have employed to support basic image processing operations including histogramming, table lookups, convolutions and morphological operations. We have developed an implementation of replicated images on the Connection Machine that is much more practical as a software solution to the problem than partitioning.

Replication can be used to address the efficient analysis of "small" instances of important data structures other than images—for example chains (which ordinarily arise from boundary extraction processes) and graphs (representing the geometric and topological relationships between elements in some segmentation of an image). Generally, the following four problems must be addressed in developing replicated data analysis for any specific data structure:

- Embedding—An embedding of the data structure into the underlying interconnection network of the machine must be identified that simultaneously optimizes two proximity criteria. First, within a single copy of the data structure, nodes that are nearby in the data structure should be mapped into processors that are nearby in the interconnection network. Second, between copies of the data structure, processors that represent corresponding nodes (e.g., pixels with the same original image address) should be nearby in the interconnection network. For images, this is easily accomplished using Gray coding methods in hypercube connected networks.

  The rationale for the first criterion is that most operations on the data structures involve movement of information between nodes in the data structure, with movement between proximate nodes more prevalent than movement between distant nodes. The rationale for the second criterion is that partial results from the copies must eventually be combined to generate a representation for the complete computation. This ordinarily involves collecting information from corresponding nodes in all copies, usually with a logarithmic merging process; this process is expedited by keeping such corresponding nodes proximate in the network.

- Distribution—Given a goal embedding, algorithms must be developed to generate the copies from the original instance of the data structure (e.g., given an arbitrary $s \times s$

window of an image, we might need to generate 16 copies of the window). Such distribution algorithms are communication intensive, and must be designed to minimize interference in the interconnection network as the copies are distributed to their destinations. Since we might want to process several replicated structures simultaneously, the distribution algorithms should, additionally, consider the problem of multiple instances and partition the machine among the instances.

- Decomposition—Methods must be developed for decomposing a computation into pieces that can be performed in parallel on the copies of the data structure. This would ordinarily be straightforward if the underlying machine architecture were MIMD. However, cellular machines are SIMD machines, and provide us with very limited flexibility in decomposing operations into pieces that can be simultaneously computed on the copies. The Connection Machine, for example, provides the capability for each processor to access a different location in an array stored in its local memory at any time. However, this location will have to be a simple function of the copy and location numbers of each node in the replicated structure, since it is unfeasible to preload the memory of the machine with a problem and data structure dependent addressing pattern. The Connection Machine also supports arbitrary interprocessor communication patterns. In replicated image processing, for example, the copy number is used to determine a relative address for the decomposition of a convolution operation. So, for example, all pixels in copy one might communicate with their north neighbor; all pixels in copy two with their northeast neighbor, etc.

- Collection—Once the projections of the operation are completed within each copy, the partial results must be collected into a final result. This ordinarily involves a logarithmic merging operation. The algorithms must be designed so that at the end of the collection stage, each copy has a complete result, as opposed to one copy being designated as the master and the remaining copies having only partial results. The reason for this is that it is likely that we will want to perform subsequent basic operations on the replicated structure, and these will proceed most efficiently if all of the copies have complete rather than partial results.

We have developed data structures and processing algorithms for two other common image data structures—chains and graphs. Chains (linear structures) arise naturally from edge detection and border following. Generally, edges comprise only a small percentage of the pixels in an image (2%-5%). Therefore, if an image has been processed and its important edges and boundaries *marked*, then processing them using conventional SIMD algorithms would be very wasteful, since most of the processors would be idle. Similarly, segmentations of an image are naturally represented using planar graphs. These graphs have far fewer nodes than there are pixels in the image, so they can also profitably be analyzed using replicated data analysis algorithms.

4.2. Multiresolution Techniques

4.2.1. Pyramids and Hough Transform on the Connection Machine

Background

Pyramid architectures and algorithms have been demonstrated to be useful in fast computation of global properties by performing only local operations. Embedded in the rapidly-decreasing sizes of the processor arrays is a multiresolution data structure which enables parallel multiresolution image analyses. Since processors at different levels in a pyramid are connected in a tree structure, the height of which is only logarithmic in the image size, global information can be extracted in logarithmic time. Log(image diameter) time performance can be achieved for image filtering and segmentation, among other tasks. We developed a pyramid programming environment on the Connection Machine. Pyramid algorithms can be implemented in this programming environment to gain more knowledge about the performance of parallel pyramid algorithms. A pyramid Hough transform program based on a merge and select strategy among line or edge segments is presented to illustrate the capability of our system.

In the Connection Machine, groups of 16 processors are placed on single chips and connected by a 4 by 4 grid. Global communication is done via a communication network called the *router*. The router is a Boolean 12-cube with every router node connected to 16 processors. In addition, there is a communication mesh called the *NEWS* network which connects the processors in a two-dimensional mesh. Local grid communication between processors is expected to be faster via the NEWS network than via the router.

For applications requiring more processors than there are in the machine, the Connection Machine provides the mechanism of *virtual processors*. If the machine has 2^K physical processors while 2^N, $K < N$, processors are needed for the application, a mechanism is created so that every physical processor simulates 2^{N-K} virtual processors, each having a unique virtual cube address N bits long. In this way, a maximum number of physical processors remain active during program execution.

Design Considerations

Our system was designed so that pyramid architectures with rectangular support from children and resolution reduction factors of 2 could be implemented easily. Children linked to a father are not limited to be only those in a 2 by 2 block but can be in any rectangular block having even sides and centered at the 2 by 2 block. Such a construct is particularly helpful in designing pyramid structures with elongated support from children as well as for overlapped pyramids. Our system also provides a programming environment compatible with other existing vision software on the Connection Machine. Users need only define the data structure of a pyramid node and the local operations desired, both inter- and intra-level, to develop specific systems. Configuring the Connection Machine into a pyramid, setting up the necessary linkages between processors, local memory allocation, and global bottom-up and top-down activation are all done automatically by our system.

Embedding Pyramids into the Connection Machine

Several methods have been proposed to map the pyramid architecture onto a hypercube architecture. Our mapping is based on a scheme called *Shuffled 2D Gray Code.* It uses *Reflexive Gray Coding.* Reflexive Gray Coding has been widely used in mapping hypercubes onto processor arrays as well as pyramid machines. It has the attractiveness that two successive codes differ by only one bit.

For the computation of the Shuffled 2D Gray Code, we first compute the reflexive Gray Code $G(i)$ and $G(j)$ of two integers i and j. We then *interleave* the bits of $G(i)$ and $G(j)$ to derive $SG(i,j)$, the Shuffled 2D Gray Code.

We identify every Connection Machine processor by its virtual hypercube address. Let *node* (i,j) represent the pyramid node on level l (the bottom level is level 0) with grid coordinates (i,j), $0 <= i,j < 2^{L-l}$, where L is the height of the pyramid. This node is mapped to the virtual processor with cube address $SG(i,j) * 4^l$. Every pyramid level therefore maps one to one onto the whole machine. The number of levels each virtual processor is involved in for this simulation is one plus half the number of trailing zeroes of its cube address. Physical processor 0 will simulate the greatest number of pyramid nodes, 13 in a 64K machine on a 512 by 512 image.

There are several advantages to using this mapping scheme. First, every pyramid level in this mapping forms a 2D Gray Code mesh. Thus the whole pyramid architecture is homogeneous. Inter- and intra-level communication involves the same operation for every pyramid node and is independent of its location.

Secondly, because each row and each column in a 2D Gray Code mesh constitutes a Gray Code sequence, each of the cube addresses of the four mesh neighbors of any pyramid node differs from that of the given node by one bit only. This means that every 4-neighbor is only one communication link away, while every 8-neighbor is only two communication links away. The communication cost between any neighboring pair of nodes on any level is greatly reduced.

Thirdly, determining parent/child relations is very easy. For processor $SG(i,j) * 4^l$, which is node (i,j) on level l, the cube address of its direct parent is $SG(i/2,j/2) * 4^{l+1}$. Computationally, this is equivalent to setting bits $2l$ and $2l+1$ of its own cube address to zero (the least significant bit is bit 0). The addresses of the four direct children, if they exist, can be determined similarly by setting bits $2l-2$ and $2l-1$ of its own address to 00, 01, 10, and 11, respectively. The direct parent of any pyramid node is, at most, two communication links away.

Our mapping scheme maps each pyramid level onto the whole Connection Machine. This is adequate for most pyramid algorithms where only one level of the pyramid is active at any time. For algorithms requiring that more than one level be active at one time, it is possible to map the whole pyramid onto the Connection Machine so that every neighbor is, at most, two communication links away, provided that the number of processors in the hypercube is at least twice the number of processors in the pyramid.

Pyramid Hough Transform

We show that a pyramid machine can be used to perform a variation of the Hough transform by implementing the following algorithm.

Every line in the X-Y plane can be represented by a pair of coordinates, (ρ, θ), where ρ is the distance from the origin to the line and θ is the directed angle from the positive X axis to the normal of the line. Every point (x, y) on this line has coordinates of the form $(\rho \cos\theta - t \sin\theta,\ \rho \sin\theta + t \cos\theta)$. The line segment from $(\rho \cos\theta - t_1 \sin\theta,\ \rho \sin\theta + t_1 \cos\theta)$ to $(\rho \cos\theta - t_2 \sin\theta,\ \rho \sin\theta + t_2 \cos\theta)$ can be represented as (ρ, θ, t_1, t_2).

Because of the exponentially tapering structure of the pyramid, information gathered from children needs to be condensed so as to be stored locally and passed up for further processing. We assume every pyramid node to have a bounded memory capacity able to store at most K 4-tuples (ρ, θ, t_1, t_2) representing line segments. Every node on level 1 collects from its four direct sons a maximum of $4K$ such tuples. It then tries to merge the line segments represented by these tuples into longer segments using a *distance* measure between segments. The K *best* segments surviving the merge are stored at that level-1 node. This process is repeated at levels 2, 3, ... until the apex of the pyramid is reached. At this stage the apex has stored the K most prominent line segments in the image.

We measured the actual spatial distances between pairs of line segments in order to merge segments which are collinear and close to one anothe . We defined the distance between two line segments as the maximum of the distances from one of the endpoints of one segment to the line on which the other segment lies. This distance is computed for all pairs of segments collected at current node. The pair with the minimal distance is merged into one segment if the distance is below a certain threshold. The process is repeated for the remaining $4K - 1$ segments until either the threshold is exceeded or the number of segments remaining equals K.

To merge two segments into one, we define the new segment as lying on the line determined by the midpoints of the two pairs of endpoints of the two segments. The endpoints of the new segment are the two outermost projections of the original four endpoints onto this line.

Finally, to decide which K segments to keep, a node computes for every merged segment the sum of the line (or edge) strengths of all pixels contributing to that segment. Shorter and weaker lines or edges will have lower strengths and will be weeded out gradually as the process continues up the pyramid. This leaves the more salient ones at higher levels in the pyramid, as desired in a Hough transform.

In conclusion, our results show that the performance of the pyramid Hough transform is not much worse than that of the ordinary Hough transform except when the globally prominent lines or edges consist of collections of collinear short segments which are not locally salient.

4.2.2. Fast Addition on the Fat Pyramid

We developed an algorithm for fast addition on a parallel fat pyramid, i.e., a fat pyramid in which each pyramid node has computational capabilities. A pyramid representation of a

$2^n * 2^n$ image is a stack of successively smaller arrays of nodes; level l of the pyramid will have a size of $2^l * 2^l$, with the root of the pyramid being level 0. In the *fat pyramid*, the size of a node depends on the level of the pyramid in which it appears. A node at level l of the fat pyramid will have four times as much storage space and processing power as a node at level $l + 1$. We assume a fat pyramid in which larger processors are implemented through the use of aggregations of smaller processors.

Our addition algorithm is based on a *carry-lookahead* technique that has been proposed in the literature. It is based on the fact that *a carry does not depend explicitly on the preceding one, but can be expressed as a function of the relevant augend and addend bits and some lower-order carry.* The technique defines two auxiliary functions: $G_i = a_i b_i$ (the carry-generate function) and $P_i = a_i + b_i$ (the carry-propagate function); G_i gets the value of 1 when a carry is generated at the i^{th} stage, while P_i gets the value of 1 when the i^{th} stage propagates the incoming carry c_i to the next stage $i + 1$. The carry c_{i+1} can then be expressed as: $c_{i+1} = a_i b_i + (a_i + b_i)c_i = G_i + P_i c_i$ which implies $c_{i+1} = G_i + \sum_{j=0}^{i-1}(\prod_{k=j+1}^{i} P_k)G_j + \prod_{k=0}^{i} P_k c_0$.

If a single processor is allocated to a single node at level n, then 4^{n-k} processors will be allocated to a single node at level k. This implies that the operands for the addition operation at level k should be distributed throughout those 4^{n-k} processors. For reasons of uniformity, the same number of bits from each operand will be allocated to each of the above processors. The addition of two N-bit operands returns an $N + 1$ bit result; however, $N + 1$ may not be perfectly divided by 4^{n-k}. This implies that sign-extension of each operand should be performed to transform it into a number having the same value as before, but with a number of bits equal to a multiple of 4^{n-k}. We will be using the *two's complement representation* of binary numbers, so if $a_{N-1}a_{N-2}\ldots a_1 a_0$ and $b_{N-1}b_{N-2}\ldots b_1 b_0$ are the binary representations of the operands A and B respectively, we will allocate a total of $4^{n-k}\lceil(N+1)/4^{n-k}\rceil$ bits to each operand and the result, where $\lceil(N+1)/4^{n-k}\rceil$ of those bits will be stored in each one of the 4^{n-k} processors. From now on, we will be using M instead of $4^{n-k}\lceil(N+1)/4^{n-k}\rceil$ (i.e., total number of bits in each extended operand) and q instead of $\lceil(N+1)/4^{n-k}\rceil$ (i.e., number of bits from each operand stored in a single processor). According to the above discussion, the sign-extension of the numbers will give us new binary representations for the operands A and B, as follows: $A = a_{M-1}a_{M-2}\ldots a_{N-1}a_{N-2}\ldots a_1 a_0$ and $B = b_{M-1}b_{M-2}\ldots b_{N-1}b_{N-2}\ldots b_1 b_0$ where $a_{M-1} = a_{M-2} = \cdots = a_N = a_{N-1}$ and $b_{M-1} = b_{M-2} = \cdots = b_N = b_{N-1}$.

The M bits of each extended operand will be divided equally among the 4^{n-k} processors, such that if (x, y) are the Cartesian coordinates of a processor in the $2^{n-k} * 2^{n-k}$ grid that corresponds to a single node at level k of the fat pyramid, then the processor will contain the bits of the operands whose subscript s satisfies the condition $s \bmod 4^{n-k} = 2^{n-k}y + x$.

The algorithm is as follows: Initially, the processor at (x, y) initializes q individual carry-in terms $ca_j(x, y)$ to zero, for $0 \leq j < q$; $ca_j(x, y)$ will represent the carry-in with subscript $j4^{n-k} + 2^{n-k}y + x$. It also initializes q individual propagate terms $Pr_j(x, y)$, that will correspond to groups preceding the bits of the operands it deals with, to 1. $Pr_j(x, y)$ will correspond to the propagate term with subscript $j4^{n-k} + 2^{n-k}y + x$, where $0 \leq j < q$.

Then, all the processors initialize in parallel q group carry and q group propagate terms for groups of size $2^0 = 1$, as follows: $Gca_j(x, y) = GG_{i,i} = a_i b_i$ and $GPr_j(x, y) = GP_{i,i} = a_i + b_i$, for $0 \leq j < q$, where $i = j4^{n-k} + 2^{n-k}y + x$. We assume that the binary representations of the Cartesian coordinates (x, y) of each processor, in the grid of size $2^{n-k} * 2^{n-k}$ it belongs to, are as follows: $x = x_{n-k-1} \dots x_1 x_0$ and $y = y_{n-k-1} \dots y_1 y_0$.

Let us denote by x'_r the binary number that differs from the binary representation of x only in bit position r, that is $x'_r = x_{n-k-1} \dots \bar{x_r} \dots x_1 x_0$ where $0 \le r < n-k$, and $\bar{x}_r$ represents the binary complement of x_r. In a similar way, let $y'_r = y_{n-k-1} \dots \bar{y_r} \dots y_1 y_0$.

Initially, pairs of processors that have the same value for the y coordinate, and whose x coordinate differs only in x_0, exchange their values of $Gca_j(x,y)$ and $GPr_j(x,y)$, where $j = 0, 1, \dots, q-1$. We consider bidirectional communication channels in the grid of processors, which means that any two neighbor processors can send data to each other at the same time.

So, the processor at position (x,y) receives $Gca_j(x'_0, y)$ and $GPr_j(x'_0, y)$, for $0 \le j < q$. Then, only the processors whose x_0's are 1 change their individual carry and propagate terms as follows: $ca_j(x,y) = ca_j(x,y) + Pr_j(x,y)Gca_j(x'_0,y)$ and $Pr_j(x,y) = Pr_j(x,y)GPr_j(x'_0,y)$ where $0 \le j < q$.

The above operations update the carry-in values and the corresponding propagate terms for the bits in those processors, since for $x_0 = 1$ we have

$$Gca_j(x'_0,y) = Gc_{j',j'}, CPr_j(x'_0,y) = GP_{j',j'}, ca_j(x,y) = c_{j'+1} \text{ and } Pr_j(x,y) = P_{j'+1},$$

where $j' = j4^{n-k} + 2^{n-k}y + x - 1$. We then have $c_{j'+1} = c_{j'+1} + P_{j'+1}Gc_{j',j'}$ and $P_{j'+1} = P_{j'+1}GP_{j',j'}$ respectively, which give updated carry-in and propagate values for the corresponding bit position, after combining consecutive bits in pairs. Then, the following operation is performed in parallel in those processors that have $x_0 = 1$, for $0 \le j < q$: $Gca_j(x,y) = Gca_j(x,y) + GPr_j(x,y)Gca_j(x'_0,y)$, otherwise (i.e., for processors that have $x_0 = 0$) $Gca_j(x,y) = Gca_j(x'_0,y) + GPr_j(x'_0,y)Gca_j(x,y)$. Also, all the processors perform in parallel the operation $GPr_j(x,y) = GPr_j(x,y)GPr_j(x'_0,y)$ for $0 \le j < q$.

The above implies $Gc_{j',j'+1} = Gc_{j'+1,j'+1} + GP_{j'+1,j'+1}Gc_{j',j'}$ and $GP_{j',j'+1} = GP_{j'+1,j'+1} GP_{j',j'}$ respectively, where $j' = j4^{n-k} + 2^{n-k}y + 2\lfloor x/2 \rfloor$. The new values of $Gca_j(x,y)$ and $GPr_j(x,y)$ will represent the group carry and the group propagate terms for pairs of consecutive bits.

Processing similar to the above continues $n-k-1$ more times. During the r^{th} execution, where $0 \le r < n-k$, any two processors that have the same value for the y coordinate but differ in the r^{th} bit of their x coordinate (i.e., their distance is 2^r in the grid of size $2^{n-k} * 2^{n-k}$) exchange their values of $Gca_j(x,y)$ and $GPr_j(x,y)$, and then perform the following operations in parallel:

if $x_r = 1$, then for $0 \le j < q$,

$$\begin{aligned} \{ca_j(x,y) &= ca_j(x,y) + Pr_j(x,y)Gca_j(x'_r,y), \\ Pr_j(x,y) &= Pr_j(x,y)GPr_j(x'_r,y), \\ Gca_j(x,y) &= Gca_j(x,y) + GPr_j(x,y)Gca_j(x'_r,y)\} \end{aligned}$$

otherwise

$$\{Gca_j(x,y) = Gca_j(x'_r,y) + GPr_j(x'_r,y)Gca_j(x,y)\}$$

and all the processors update the group propagate terms as follows:

$$GPr_j(x,y) = GPr_j(x,y)GPr_j(x'_r,y).$$

Just after the r^{th} execution of the above steps, where $0 \leq r < n - k$, the processor at (x, y) will contain the group carry (assuming a carry-in of 0 for the group at this point) and the group propagate terms for the q groups of 2^{r+1} consecutive bits each, that are stored in the processors with the same y coordinate and whose x coordinate in the $2^{n-k} * 2^{n-k}$ grid is in the range from $2^r\lfloor x/2^r \rfloor$ to $2^r\lfloor x/2^r \rfloor + 2^r - 1$. Also, the final value of the carry-in for the bit positions 0 through $2^r - 1$ will have been computed.

After those $n - k$ "cycles", the processor at (x, y) will contain the $2q$ auxiliary group terms for the q groups of bits $j4^{n-k} + 2^{n-k}y$ through $j4^{n-k} + 2^{n-k}(y + 1) - 1$, for $0 \leq j < q$ (i.e., the combination of data contained in all the processors whose second coordinate is y). Similar types of operations are then performed for the y dimension. Groups of bits are combined using only vertical communication patterns in the grid.

So, to summarize, communications and computations occur during the first $2(n - k) = \log p$ "cycles" of the algorithm, where $p = 4^{n-k}$ is the number of processors allocated to a single node at level k of the pyramid, while only computations occur during the remaining q "cycles." The computation time of the algorithm is proportional to $\log p + q$, if we assume that the system contains m-bit processors, where $m \geq q$. If we consider 1-bit processors, then the computation time of the algorithm will be proportional to $q * \log p$, because each of the operations presented before will be executed q times in each "cycle" of the algorithm.

4.2.3. Replicated Image Processing

The conventional way of mapping images onto large SIMD machines like the Connection Machine, that of assigning a processing element per pixel, tends to let major portions of the machine lie idle, especially when the image is relatively small, as is the case in focus-of-attention image processing. We developed a method to remedy this situation, gaining on the processing time as a result. We replicate the image as many times as possible, and let each copy solve a part of the problem. The partial results from the individual copies are then aggregated to get the solution to the problem. We call this method replicated image processing. We developed replicated image algorithms for histogramming, table lookup, and convolution on the Connection Machine and compared their performance with the non-replicated algorithms for the same.

In a replicated pyramid, the cells in the hypercube are used to replicate, as many times as possible, the image stored at any given level of the pyramid. So, for example, if the image at the base (the 0^{th} level) of the pyramid has exactly as many nodes as there are cells in the hypercube, then only one copy of the image would be stored in the hypercube. At the first level there would be sufficient cells to store four copies of the reduced resolution image. In general, at the l^{th} level we would have $4l$ copies of the reduced resolution image. It is straightforward to embed the replicated reduced resolution images into the hypercube using Gray codes.

We have implemented the histogramming, table lookup, and convolution algorithms for a replicated pyramid architecture. A single level of the replicated pyramid was implemented on the Connection Machine by configuring it as an $n \times m \times m$ array of processors, where n is the number of copies of the $m \times m$ image. Communication is efficient this way, since the Connection Machine automatically assigns subhypercubes to each copy of the image.

Communication within each copy remains local and is independent of the communication within other copies. Also, the logarithmic merging step, typical of the second phase of all three algorithms is very fast because corresponding elements of different copies of the image are adjacent in the Connection Machine hypercube.

A copy of the histogram of the image (in the case of the histogramming algorithm) or the entire table (in the case of the table lookup algorithm) was stored in each copy of the image. Histogramming and table lookup distribute the problem among the $4l$ copies available in such a way that each copy handles an equal number of gray-level values of the entire image. The implementations of these two algorithms are very similar, so we discuss only the histogramming algorithm. The histogramming algorithm takes advantage of the CM2 Connection Machine feature that if each processor sends a message to the processor that counts its gray-level value, then there will be as many collisions at each counting processor as the number of pixels with that gray-level value. Using the collision resolution strategy of counting all the colliding values, the histogram for the entire image can be computed in a single step. Thus, the histogramming algorithm implemented on a single level of a replicated pyramid cannot outperform a single copy histogram algorithm on the Connection Machine.

The convolution algorithm yields more interesting results on replicated pyramids. To compute a $k \times k$ convolution, k^2 copies of the image are required. (On the Connection Machine, the next power of 2 has to be chosen as the actual number of copies allocated, although only k^2 of them are used.) Each copy handles one of the kernel weights. In the first phase, each processor multiplies this kernel weight with its own gray-level value and sends the product to the appropriate processor in the same copy. In the second phase, these products are summed to obtain the final result of the convolution at every processor of every copy. We compared the time taken by the convolution algorithm implemented on the CM using a single copy against the replicated algorithm using k^2 copies. Experiments indicate that the utilization of the Connection Machine is much higher for the replicated algorithm than for the single copy algorithm. This can be explained as follows. If there are a sufficient number of physical processors to store all k^2 image copies, then the replicated algorithm performs one multiplication step (in which all k^2 kernel multiplications are performed in parallel) and $2 \log k$ addition steps to compute the multi-copy convolution. In contrast, the single copy convolution algorithm has to perform k^2 steps of multiplications and additions, all under direction of the host computer. Thus, the host overhead is much higher for the single copy algorithm than for the replicated algorithm.

There are many basic image analysis algorithms to which the methods described above can be extended in a straightforward way. For example, gray scale morphological operations are implemented in much the same way as gray scale convolution; various types of image statistics useful for image texture analysis, such as co-occurrence matrices or difference histograms, can be computed very quickly also.

4.3. Quadtrees on the Connection Machine

We developed a general technique for creating SIMD parallel algorithms on pointer-based quadtrees. It is useful for creating parallel quadtree algorithms which run in time proportional to the height of the quadtrees involved but are independent of the number of objects (regions, points, segments, etc.) which the quadtrees represent, as well as the total number of nodes. The technique makes use of a dynamic relationship between processors and the elements of the space domain and object domain being processed.

Consider the task of constructing a quadtree for line segment data (a *PM quadtree*). In constructing a PM quadtree, a node should be assigned the color gray and subdivided if its boundary contains more than one endpoint, or if its boundary has two segments which enter it but which do not have a common endpoint within it. Initially, we have one processor allocated for the quadtree root, and one processor for each line segment, containing the coordinates of the segment's endpoints.

Consider creating an algorithm, to construct the PM quadtree for this segment data. Each segment processor initially possesses a pointer to the quadtree root processor. Each segment processor computes how many of its segment's endpoints lie within the boundary of the node to which the segment processor points; this will be 0, 1, or 2. Each segment then sends this value to the node it points to, and both the maximum and minimum of these values is computed at the node. Any node which receives a maximum value of 2 assigns itself the color gray, since this means that some single segment has both endpoints in the node's boundary. Any node which receives a maximum of 1 and a minimum of 0 also assigns itself the color gray, since this means that there are at least two segments in the node's boundary, one which passes completely through it and one which terminates within it.

Each segment with exactly one endpoint in the node it points to then sends the coordinates of that endpoint to the node. The node receives the minimal bounding box of the coordinates sent to it (this, of course, amounts simply to applying min and max operations appropriately to the coordinate components). If this minimal bounding box is larger than a point, the node assigns itself the color gray, since this means that some two segments entering the node have non-coincidental endpoints within the node.

Finally, each segment with no endpoints in the node it points to determines whether it in fact passes through the interior of the node at all; if so, it sends the value "1" to the node, where these values are summed. If the sum received by the node is greater than 1, the node assigns itself the color gray, since this means that some two segments passing through the node do not have any endpoints in the node, which implies that they do not have a common endpoint in the node. Then all gray nodes allocate son processors. Any nodes which were not given the color gray should be colored white if no segments entered their interior (the sum is zero), and black otherwise (the sum is one).

At this point in the algorithm, we would like to have all segment processors which point to gray nodes compute which of the node's sons they belong to, and retrieve from the node the appropriate son pointer. Of course a given segment can intersect more than one of the node's sons, and we are left with the situation of wanting to assign up to four son pointers to the segment processor's node pointer, and processing each of the corresponding sons. The solution to this dilemma is to allocate *clones* of each such segment processor, that is, to create multiple processors which represent the same segment, and all of which contain (almost) the same information. So for each segment processor pointing to a gray node, we allocate three clone processors, all of which contain the segment's endpoints and a pointer to the same node as the original segment processor. In addition, the original and its clones each contain a *clone index* from 0 to 3, with the original containing 0 and each of the clones containing a distinct index from 1 to 3. Now the original and its clones each fetch a son pointer from the node that they all point to, each one fetching according to its clone index, so that each gets a different son pointer.

The subsequent iterations of the algorithm proceed as the first, with each segment processor determining how many of its endpoints lie within the interior of the node it points to, and with the eventual computation of the colors of all the nodes on each particular level. At this point in each iteration, notice that any segment processors pointing to leaf nodes, or whose segments do not pass at all through the interior of the node to which they point, will not have any further effect on those nodes, and thus be de-allocated and re-used later. This reclaiming of segment processors keeps the number of clones allocated for each segment from growing exponentially. In fact, the number of processors required for a given segment (at a given level in the construction of the quadtree) will be roughly the same as the number of nodes (at that level of the tree) through whose interiors the segment passes.

To summarize the general technique, we allow one processor per quadtree node, and initially allow one processor per object. Each object is given access to a sequence of shrinking nodes which contain part of it; initially all objects have access to the root node. By having each object obtain information from its node, and by combining at each node information from all of the objects which access that node, the objects make decisions about descending the quadtree from that node. For those objects which do descend, it is desirable for their various parts which lie in various quadrants of the node to descend in parallel. Thus, we allow duplicate or 'clone' processors for each object, and have each processor handle just that portion of the object relevant to one quadrant of the node. Duplicate processors which determine that they can no longer affect the node to which they point, because that node is a leaf, or because the object they represent does not overlap that node, can deactivate themselves so that they may be used later in the computations for some other object.

We see then that this technique allows us to go beyond the level of granularity of one processor for every element (space component or object) to a level where there are multiple processors for certain elements and none for others; where the processors are being used and disposed in a dynamic fashion.

The same general technique can be applied to create algorithms for several other quadtree tasks, such as shifting, rotation, and expansion, which run in time proportional to the height of the new quadtree, by computing in the parallel the rotated or expanded version of each old black leaf node, and building the new quadtree using cloning.

4.4. Benchmarking Activities

4.4.1. Border Tracking with an Implementation on the Butterfly Parallel Processor

Background

In digital image analysis, objects visible in an image can be recognized by describing their borders, and then matching the border representations to model descriptions. There are basically two ways of tracking the borders of a given object. The first is studying local gray level values in a small neighborhood, and producing a strong response for fast gray level changes at object edges and a low response for image locations with constant or slowly varying spatial gray level distribution. The strongest edges are separated from the weaker ones by thresholding the gradient magnitude image, exploiting for example the edge strength histogram. The remaining borders are finally followed to produce coordinate lists corresponding to the 4- or 8-connected edge point patterns in the image. The other alternative differs from the first, in that the segmentation phase preceding the border following is not trying to locate the edges of an object directly by studying local gray level changes, but by first locating the object body and then tracing its outer border and the inner borders corresponding to the holes in the body. In this work an algorithm based on the second approach was developed.

The real-time operation requirement of complicated vision systems necessitates the use of fast parallel implementations of image analysis algorithms. In particular low and intermediate level operations require processing of large amounts of input data, and are potential objects for parallelization. In this study, a parallel version of the border tracking algorithm was developed for a multiprocessor system classified as a MIMD computer. The specific computer used in testing the algorithm was the Butterfly Parallel Processor with an 88 processor configuration.

Sequential Algorithm for Border Tracking

The algorithm tracks the borders of an area-segmented binary image. The input image is processed by one pass in a row-by-row fashion, starting on the highest row and proceeding downwards. Instead of outputting edge point coordinates, a crack code representation of the border shape is used. The crack code of an object border is produced in the following way: if we follow the cracks around a border, at each move we are going either left, right, up or down; if we denote the direction 90^*i by i, these moves can be represented by a sequence of 2-bit numbers $(0,1,2,3)$.

There are two kinds of borders in an image: outer and inner borders. An outer border encloses an object, whereas an inner one surrounds a hole in an object. In this algorithm the outer boundaries are followed in a clockwise order and the inner ones in a counterclockwise order. The opposite directions result from the principle of tracing the edges in such a way that the object is always kept on the right side of the trace path. The objects are taken as 4-connected and the background as 8-connected.

An image is processed by moving a 2×2 window through the image in a raster scan fashion: each row from left to right, row by row from top to bottom. The actions to be performed depend on the neighborhood visible in the window, resulting in sixteen primary operations on the crack code strings. The primary operations resolve themselves into various subactions, such as extending a code string head or tail, merging of two strings, and creation of a new outer or inner string.

A 2×2 window doesn't always contain enough information to carry out certain string operations. An example of such a case is the presence of neighboring runs on the previous row, which often lead to the merging of two strings, or the creation of a new inner or outer string. In this algorithm a knowledge of the runs yet to come on the current row is not necessary, even if those runs turn out to be neighbors to the current run. Every decision is made using only the local history of the row and the contents of the current window. The local history of the row contains two pieces of information. The first one is a flag which indicates whether there is a neighboring run in the previous row which terminates before the current position on the row. If there is, the flag points to the right end of the neighboring run. A similar flag is maintained for the current row. When the right end of the current run is encountered, the flag is set to point to the end column if there is a neighboring run on the previous row which still continues. The presence of a neighboring run on the previous row sometimes results in the merging of two code strings or to the creation of a new inner string.

The algorithm contains two important data structures. The first is called an 'active string list' and the other one is called a 'modified string list'. While scanning a row, the active string list contains the addresses of the active strings for the current row, i.e., strings which have not yet been updated on this row. After updating a string it is moved to the modified string list. The modified string list becomes the active string list on the next row and a new empty modified list is then created. As a string becomes closed, it is removed from the lists and output.

Parallel Algorithm for Border Tracking

Parallelization of this kind of sequential algorithm seems quite straightforward. The input image is divided into a given number of equal sized blocks overlapping each other by one row, each of them is processed separately, and finally the partial results are combined. This leads to a two-stage algorithm where the second stage cannot be started before the first one is completed. The parallelization of the first part is obvious, but the necessity for parallelism in the second stage depends on the amount of partially completed data produced in the first stage. In this application the input aggregate for the merging process depends on input image size and its contents and especially on the number of image blocks. The second stage was found to be the bottleneck for the performance in several experiments, which motivated us to design a parallel algorithm for merging of partial code strings, too.

There is an important point to be considered with the partition of image data, though. The easiest way of doing the partitioning is to assign an equal number of rows to each processor. This is the first approximation of equal work load for the processors. But a simple principle like this often leads to improper balancing among the processors if the contents of the image is not evenly distributed. Because the execution time of a parallel program step is dominated by the slowest parallel subtask, it is important to do the division of input data by exploiting a better estimation for the quantity of work.

The main modification to the sequential algorithm is due to the possibility that an object border may traverse several image blocks. For this reason, the top row and bottom row of a block contribute to extra actions in many of the primary cases. To overcome these implications a few additional members must be included in the code string data structure. The most important of these are binary flags to indicate lower or higher border crossings for string head and tail components. Also, the intersection coordinates must be recorded in the structure. These additional data are used in the merging stage for deciding on connectedness of partial strings in neighboring image blocks.

Classification of strings into outer and inner borders becomes somewhat meaningless within an image block if they traverse two or more partitions, because just by looking at a partial string it is not possible to make the distinction. These attributes are useless in the merging stage, too. If a string can be completed in a block, though, its classification holds. There is a simple way to determine reliably whether a completed string is an outer one or an inner one. This can be carried out by studying the local neighborhood of the highest border point of the string, called the reference point of the string. It is the first point of the associated border that is found during scanning the image. The special meaning of reference point is due to the action of creating a new partial string at this location. Because borders are always followed in the same direction (keeping the object on the right), the crack codes emanating from the reference point are different for an outer string and an inner string. The crack code sequence for an outer string in the vicinity of the reference point, coming along the head and continuing along the tail, is '...1-0...'. The sequence for an inner string is respectively '...2-3...'. During the border tracking stage, each processor assigns a reference point to every string in its block. If a string crosses the upper limit of the block, the intersection point is also the local reference point of the partial string. After the merging step, the global reference point is searched for along the combined string chain and the classification is finally done.

Conclusions

A general algorithm for extracting object borders was developed. The binary image is processed in a raster scan fashion in one pass, producing crack code strings describing the borders. The objects are regarded as 4-connected, but an 8-connected version is easily achieved. The algorithm can be generalized to process gray level images, too.

A two-stage parallel version of the algorithm was developed, where the input image is partitioned into partially overlapping equal-sized blocks, each of which is processed by a separate processor. The first step produces crack code descriptions of the object borders hitting the blocks, in parallel. The second step is required for merging the incomplete border strings traversing more than one image block.

Our experiments on the Butterfly Parallel Processor reveal that with an image size of 512 by 512, the speedup increases nearly linearly up to the point where about twenty processors are in use. After that, the contention for shared memory resources starts leveling off the performance growth severely. One important source for the increase in execution time after this critical point is the merging step which is more and more heavily loaded as the number of incomplete border descriptions increases with the increased number of image partitions. The merging step can be omitted from the algorithm.

4.4.2. Parallel Matching of Attributed Relational Graphs

Background

In many computer vision applications, it is necessary to utilize structural analysis in pattern classification. To facilitate this, an appropriate structural description of objects and their mutual relations is required. In addition, a fast classifier for these descriptions should be available.

Attributed relational graph description has been found to be very suitable for computer vision. There are several approaches to match the graphs. These algorithms vary from complex syntactic methods to simple matching techniques. Unfortunately, the optimal comparison of two graphs is inherently an NP-complete problem. Therefore, the computational efforts required increase exponentially with the number of nodes in the graphs. For example, the matching of two k-node graphs using conventional algorithms may require the construction and evaluation of $k! * k!$ solution candidates in the worst case. This means that when using conventional computers and algorithms the optimal comparison of graphs with more than a few nodes may take too long for practical applications.

In most applications, however, some physical and heuristic constraints can be applied to slow down the exponential explosion. Furthermore, the matching can be parallelized to achieve almost a linear speedup in a multi-processor environment. Finally, the entire structural classifier, based on graph matching, can be parallelized to match the graph to all the models simultaneously.

We describe a fast graph matching algorithm and its parallel implementation on two MIMD computers: a Butterfly Parallel Processor, and a Transputer system.

A Fast Graph Matching Algorithm

In our earlier work, a fast matching algorithm was developed for structural classification. The algorithm can be used to solve both isomorphic and monomorphic problems. The resulting numeric value describes the edit distance between the graphs. The algorithm has been proved to be useful for defect classification in visual inspection applications.

The edit distance is defined as the minimum number of changes to make the graphs similar to each other. The changes allowed are the deletion and the addition of a node, a link, or an attribute. The structural classifier calculates the edit distance between the graph to be classified and the model graphs representing different classes. The graph is classified in the class for which the minimum distance was found.

The basic problem is to find the best possible mapping between the nodes of the graphs to be matched. To accomplish this, a depth-first search is used to build a state space tree of solution candidates. When the search terminates, the resulting candidates are described by paths from the root to the leaves. In the worst case, the tree contains all the combinations of the two sets of nodes corresponding to the graphs. For this reason, heuristic constraints are applied to pruning the tree in order to achieve sufficient speed.

The first heuristic for limiting the size of the tree is using a certain predetermined model node to align the matching process. This base node represents the most important object or part of the object in the model description. If the problem graph contains the base node, a partial match of the graphs has been found, and only one out of k main branches of the tree remains for further study. The second heuristic exploits the local structural similarities of the graphs. The remaining $k-1$ nodes are matched level by level in the graphs starting on the successor levels of the base nodes. The resulting state space tree often consists of only a few paths.

This ordered matching enables a speedup of several orders of magnitude without sacrificing recognition accuracy, because all feasible mappings between the graphs are still considered.

The best match is found by evaluating each candidate and choosing the one which represents the minimum edit distance. In the case of monomorphism, empty (NIL) nodes are ignored in the evaluation. The evaluation time is minimized by using lookup tables for edit distances between node and link pairs.

Implemented Algorithm

The original algorithm was implemented on a Symbolics 3645 which supports list processing by hardware. MIMD-classified multiprocessing systems usually have no special processors tailored for symbolic data processing but are rather designed for numerical calculations. To better utilize the computational power of our target computers, all the symbols (node attributes, link attributes etc.) are hashed into integer values. The method for constructing the state space in a depth-first manner is changed to a breadth-first type algorithm, and the evaluation process is sped up by including several auxiliary tables in the algorithm.

A Parallel Algorithm

For a given number of processors in a MIMD computer, the most effective parallelization is reached by parallelizing the outermost loop in the program. In this way, the speedup is nearly a linear function of the number of processors if there is no interaction between the loops. The degree of sublinearity of a speedup curve is directly affected by the communication overhead between parallel processes. The generation of a state space can be distributed effectively to processors in such a way that every processor constructs its own main branch of the tree, which is the outermost loop in the sequential program. This gives an asymptotic speedup of $O(k)$, where k stands for the size of the model graph counted in nodes. More speed can be achieved by parallelizing the process deeper in the tree. For example, if parallelization is done one level deeper there will be $k-1$ processors for each main branch. Thus the speedup is $O(k*(k-1)) = O(k*k)$. This has the side effect that the number of processors increases very quickly as the graph size is increased.

For practical applications, a compromise between speed and system size has to be made. In this work, the process of state space generation is parallelized on the highest level of the tree, which yields a speedup of $O(k)$. Identification of the base node in the problem graph reduces the size of the tree to one main branch, in which case the parallelization is carried out on the next highest level. The speedup here is $O(k-1) = O(k)$ compared to the respective sequential version.

Parallelization of the state space evaluation is done in the same way by dividing the tree into a given number of equal-sized subtrees.

Results

The parallelization of the algorithm was designed especially to suit MIMD computers, like the Butterfly Parallel Processor and the Transputer network used in our experiments. The Transputer-based multiprocessor system was used to study the properties of our algorithm.

Butterfly experiments were also performed to compare the performance of our system to a commercially available multiprocessor. The results show that the program runs two to three times faster in the Transputer system than in the BPP. The performance ratio, however, strongly depends on the system clock frequencies and the speed of the IC components selected for the hardware implementation of these target computers. Programming languages were also different (C with the BPP and Occam with Transputers), which contributes to comparison difficulties.

With the graph sizes of three to seven, the speedup of the parallel algorithm is nearly linear. The increasing deviation from a linear speedup with increasing graph size results from the more extensive communication overhead, due to longer messages describing the properties of the graphs. With larger graph sizes, our Transputer system runs out of processors if no modifications to the algorithm are done. If there are M branches to be processed with the maximum of N processors, the generation and the evaluation of the state space must be done in $L = |M/N|$ consecutive steps. The execution time will thus be $L * T$, where T is the processing time for N branches.

5. Parallel Iterative A* Search

5.1. Overview

We have developed a distributed best-first heuristic search algorithm, *Parallel Iterative A^** (*PIA^**). We have shown that the algorithm is admissible, and have given an informal analysis of its load balancing, scalability and speedup. To empirically test the PIA^* algorithm, a flow-shop scheduling problem has been implemented on the BBN Butterfly Multicomputer using up to 80 processors. From our experiments, the algorithm is capable of achieving almost linear speedup on a large number of processors with relatively small problem size.

5.2. The A* Algorithm

The A^* search procedure can be described using graph-theoretical terms. For any problem instance, a state-space graph is implicitly defined. Each node in the state-space graph represents a state. As A^* proceeds from a start node s, nodes in the state-space graph are gradually expanded by a node expansion operator.

A^* finds an optimal cost path from the start node, s, to a set of goal nodes. In A^*, a node n is assigned an *additive cost*, $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost of reaching node n from s, and $h(n)$ is the heuristic or estimated cost of reaching a goal node from node n. A variant of the A^* search procedure which does not test for duplicate nodes is provided below:

1. Put the start node s on a list called OPEN.
2. If OPEN is empty, exit with failure.
3. Remove from OPEN a node n whose f value is minimum.
4. If n is a goal node, exit successfully with a solution.
5. Expand node n, generating all successors that are not ancestors of n and adding them to OPEN.
6. Go to step 2.

The omission of the check for duplicate nodes is appropriate if the state-space graph is a tree. For a graph search, there is a tradeoff between the computational cost of testing for duplicate nodes and that of generating a larger search tree. The computation required for identifying duplicate nodes can be quite substantial, especially in a distributed computational environment. In the following discussion, we assume it is worthwhile to omit the test for duplicate nodes.

5.3. The PIA* Algorithm

*PIA** proceeds by repetitive synchronized iterations. At each iteration, processors are synchronized twice to carry out two different procedures: the ***node expansion procedure*** and the ***node transfer procedure.*** Operations are largely local to the processor in the node expansion procedure and are completely local in the node transfer procedure. Data structures in *PIA** are distributed to avoid bottlenecks. Node selection, node expansion, node ordering and successor distribution operations are fully parallelized. Processors performing searches which are not following the current best heuristics are synchronized to stop as soon as possible to reduce search overhead (the increase in the number of nodes that must be expanded owing to the introduction of parallelism). During processor synchronization, speculative computations are contingently performed at each processor, trying to keep processors always productively busy, to reduce synchronization overhead. Unnecessary communications are avoided as long as processors are performing worthwhile search to reduce communication overhead. A symmetric successor node distribution method is used to control load balancing. Finally, the correct termination of *PIA** is established by its iterative structure.

The parallel architecture model we assume consists of a set of processor-memory pairs which communicate through an unspecified communication channel. The communication channel can be realized using a shared memory or by message passing. Memory referencing through local memory is completed in constant unit time. A remote reference through the communication channel, however, requires $O(\log P)$ time in the worst case with no conflicts, where P is the number of processors. Note that this architecture model is general enough to subsume most scalable multicomputers which are currently available commercially.

5.3.1. Data Structures

In each processor j, two lists are maintained in its local memory: the work list (WL_j) and the reception list (RL_j). WL_j is a priority queue and RL_j is a simple list. At the beginning of each iteration, WL_j contains the sorted nodes awaiting expansion by processor j, and $RL_j = \emptyset$. During the node expansion procedure, successor nodes generated are distributed to RL_j, $j = 0 \cdots P-1$, using a successor distribution algorithm to be described below.

Henceforth, we will use WL_j^i to denote the set of nodes in the work list of processor j at the beginning of iteration i. If the subscript is omitted, it means the union of all P processors' work lists. That is,

$$WL^i = \bigcup_{j=0,\dots,P-1} WL_j^i.$$

If the superscript is omitted, it means for all iterations. Similar notation will be used throughout this section.

Initially, $WL^0 = s$ and $RL^0 = \emptyset$. As *PIA** proceeds, WL^i is similar to a snapshot of the OPEN list in A^*, but distributed.

5.3.2. Iteration Threshold, Mandatory Nodes and Speculative Nodes

A threshold t^i is associated with each iteration i, with $t^0 = h(s)$. At each iteration, *all* nodes in WL^i with $f(n) \leq t^i$ will be expanded, and *some* nodes with $f(n) > t^i$ will be expanded.

A node $n \in WL^i$ is thus called a *mandatory node* if $f(n) \leq t^i$. Let M^i, which is a subset of WL^i, be the set of all mandatory nodes. As we shall see later, all mandatory nodes will eventually be selected for expansion by either A^* or PIA^* in the worst case.

A node $n \in WL^i$ is defined to be a *speculative node* if $f(n) > t^i$. Let S^i be the set of all speculative nodes. Then, $WL^i = M^i \cup S^i$.

Initially, $M^0 = s$ and $S^0 = \emptyset$.

5.3.3. The Node Expansion Procedure

The node expansion procedure at each iteration i operates as follows. A processor j first expands all the nodes from M_j^i and, by an algorithm to be described below, puts all the successor nodes generated into the reception lists (RL). Then, as long as any other processor is expanding a mandatory node, this processor continues to expand the *best* speculative node from S_j^i. When all the nodes from M^i have been expanded, all processors synchronize for the node transfer procedure which is described below.

In PIA^*, successors generated by a node expansion are not considered for expansion until the next iteration; they are added to RL immediately after they are generated.

5.3.4. Determining t$^{(i)}$

The threshold for the discrimination of mandatory nodes and speculative nodes at iteration $i+1$, $t(i+1)$, is defined as the maximum of:

(a) t^i, and

(b) the minimum cost of:

  (b$_i$) all successors generated from nodes in M^i, and

  (b$_{ii}$) the nodes in S^i.

There is a simple, efficient parallel method for computing t^{i+1}. For each processor j, a local constant c_j, which is the minimum cost of (a) all successors generated from nodes in M_j^i, and (b) the best node in S_j^i, can be computed during the node expansion procedure. Then, the minimum of c_j, $j = 0 \cdots P-1$, can be computed in parallel in time $O(\log P)$ while processors are synchronized; t^{i+1} is then the maximum of t^i and the computed minimum.

5.3.5. The Successor Distribution Algorithm

Successor nodes generated by a processor are put into RL_j, $j = 0 \cdots P-1$, in a *multiplex round-robin* fashion. More precisely, suppose that the most recent successor node generated by processor j is added to RL_i. Then the next successor node generated by processor j will be added to RL_k, where $k = (i+1 \pmod P)$. At each iteration, processor j sends its first generated successor to RL_j.

The advantage of this approach is that it is simple to implement and its symmetric structure helps PIA^* attain the desired load balancing. Since the successors generated are not considered for expansion until the next iteration, some optimization can be made for message-passing architectures. Messages for successor distribution can be asynchronous so that computation and communication can be overlapped. For architectures which require large communication setup time, successor nodes generated can be distributed and cached in local memory and not sent until an efficient message size for the underlying architecture is reached.

5.3.6. The Node Transfer Procedure

After the node expansion procedure, each processor j empties the nodes from RL_j and inserts them into WL_j to form a new priority queue for the next iteration. Note that the node transfer procedure is completely local; no communications between processors are required.

5.3.7. Termination

When a mandatory node is found to be a goal node by a processor, a message can be broadcast to inform all processors to terminate. If a speculative node is found to be a goal node, this node is simply added to RL because it may not be an optimal goal node.

PIA^* can terminate, failing to reach a goal node, when $WL^i = \emptyset$. $WL^i = \emptyset$ if and only if $WL^i_j = \emptyset$, for all $j = 0 \cdots P-1$. This state can be recognized and broadcast to all processors at the end of the node transfer procedure.

5.4. Analysis

The algorithm can be proven to be *admissible*, meaning that if a goal state is reachable then an optimal goal will be reached. Informal analyses and experiments have shown that the algorithm maintains good load balancing among the processors and can achieve a nearly linear parallel speedup.

6. Parallel Matrix Operations

6.1. Generalized Matrix Inversion on the Connection Machine

We have also investigated the practical implementation of algorithms for generalized inversion (g-inversion) of a matrix on the Connection Machine.

The Connection Machine is extremely well suited to data level parallelism. Accordingly the suitable algorithms are those which are deterministic and exhibit massive data parallelism. While there are several numerical algorithms available for matrix inversion, we recommend the Ben-Israel-Greville algorithm as it has the following distinct advantages:

i. Deterministic
The procedure involves only matrix multiplications and the initial approximation can be chosen to ensure convergence deterministically.

ii. Reliable
If the matrix is singular or rectangular, the least-squares or Moore-Penrose inverse is obtained, so the process does not fail.

iii. Stable
The algorithm is self-correcting and stable, and permits the use of coarse precision at earlier stages and finer precision towards the end.

iv. Linear time
Since the algorithm is entirely based upon repetitive parallel matrix multiplication (which takes linear time), it generates the inverse in linear time.

v. Scalable
The basic steps in this algorithm allow extension to larger matrices by using the virtual processor configuration on the Connection Machine. Here each physical processor can simulate a two-dimensional grid of virtual processors. In such a case, the speed of each processor is reduced by a factor of V/P, where
V = total number of virtual processors
and P = total number of physical processors.

The complexity of matrix-partitioning schemes for the g-inversion on the Connection machine has also been analyzed. It turns out that the use of the virtual processor configuration on the Connection Machine is of comparable efficiency to using any partitioning scheme, when the multiplicative iterative scheme is used for g-inversion.

6.2. Grid Evaluation-Interpolation on the Connection Machine

6.2.1. Grid Evaluation-Interpolation using Tensor Products and General Inversion

The interpolation and approximation of functions of two or more independent variables have recently become important because of their extensive technical applications in a wide range of fields—digital image processing, digital filter design, topography, photogrammetry, geodesy, and optical flow, to mention only a few. In all these applications it is required to construct formulae that can be efficiently evaluated.

Tensor products are widely used in the evaluation and interpretation of functions as well as 2D and 3D image blocks. We implemented a tensor product on the Connection Machine. We have developed a set of ready to use tensor product approximation schemes for data arrays of size (2×2), (3×3) and (4×4) in 2D and 3D. We use bilinear, trilinear and higher order forms for this purpose. These schemes can be used recursively for larger array sizes on the Connection Machine. The tensor product approximation is computationally economical to implement. For instance, in the 3D case for the $(n \times n)$ grid, we need to precompute and store only the inverses of three matrices of size $(n \times n)$ rather than one of size $(n^3 \times n^3)$. Obviously, error is introduced in such an approximation; Gordon and Schumaker provide explicit error bounds for this, as well as other related approximation schemes.

6.2.2. Grid Evaluation-Interpolation using Multivariable Spline-Blending Approximation

Instead of using polynomials $(1, x, x^2, \ldots)$, one can use Lagrange or spline functions for interpolation and use blending approximation that combines the approximations obtained using coarse and fine grids. Computationally, one of the most convenient classes of methods for this purpose is the "product-operator method," where the approximating function is calculated by treating the individual variables separately. A particular scheme is the projection operator technique of Gordon, which uses the cardinal splines, and provides a substantial saving in the number of function values that are required to approximate a function to a prescribed accuracy.

We have developed a Connection Machine implementation of the projection operator technique for multivariable cardinal spline interpolation. For this purpose we have implemented several data-parallel operations such as inner product and tensor product of vectors (whose components are single variable polynomials). These give rise to the functional form of approximation; hence we can perform symbolic differentiation and integration of the approximating functions directly. The technique uses orthogonal polynomial basis functions and their tensor products without requiring matrix inversion.

7. Conclusions

Our research program has focused on two inter-related goals:

1. The identification of new representations and algorithms for autonomous navigation that address fundamental problems in both vision and planning.
2. The development and integration of these algorithms on m[illegible]y parallel computers.

Our research on navigation has focused on problems relating to the detection, pursuit and interaction with a three dimensional model of known geometry. Specific vision problems addressed included parallel algorithms for contour analysis for the purpose of identifying feature points in images of the target, parallel algorithms for instantaneous pose estimation of the target, and Kalman filtering algorithms for developing models for the motion of the target through space and for generating trajectories toward the target that minimize the risks of collision with the target.

Turning to parallel processing, our research has uncovered several fundamental problems in the effective application of massively parallel computing to vision and planning. First, the importance of both multiresolution image analysis and focus-of-attention vision has led us to consider various methods for efficiently processing small images and other data structures on massively parallel computers. Our research has addressed the efficient processing of image arrays and contours. Second, the need to plan effectively in dynamic environments has led us to develop trajectory planning algorithms both for RAMBO and for ground vehicles moving over terrain modeled by digital terrain models.

8. Report Abstracts and Summaries

This section presents the abstracts of the 45 research reports published under the contract. For many of these reports, details have been given in the previous sections. However, over the course of the project, Ph.D. students and visiting researchers have also written reports which explore many other facets of the subject, and it was difficult to group the account of their work into a few stand-alone sections, because of the lack of a common theme in the research directions. In these cases we have supplemented the abstracts of the reports with extended summaries when the abstracts did not adequately describe the scope of the research.

1. D. DeMenthon, "Reconstruction of a Road by Matching Edge Points in the Road Image", CAR-TR-368, CS-TR-2055, June 1988

ABSTRACT: A method for the reconstruction of a road in 3D space from a single image is presented. The world road is modelled as a space ribbon generated by a centerline spine and horizontal *cross-segments* of constant length (the *road width*) cutting the spine at their midpoints and normal to the spine. The tangents to the road edges at the end points of cross-segments are also assumed to be approximately parallel. These added constraints are used to find pairs of points (*matching points*) which are images of the end points of world cross-segments. Given a point on one road image edge, the proposed method finds the matching point(s) on the other road image edge. Surprisingly, for images of road turns, a point on one road image edge has generally more than one matching point on the other edge. The extra points belong to "ghost roads" whose images are tangent to the given road image at these matching points.

Once pairs of matching points are found in the image, the reconstruction of the corresponding world cross-segments is straightforward since cross-segments are assumed to be horizontal and to have a known length. Ghost road cross-segments are discarded by a dynamic programming technique. A benchmark using synthetic roads is applied, and the sensitivity of the road reconstruction to variations in width and bank of the actual world road is evaluated and compared to the sensitivity of two other algorithms. Experiments with a sequence of actual road images as the Autonomous Land Vehicle (ALV) moves down a road are also presented.

2. T. Seppänen, T. Westman and M. Pietikäinen, "Parallel Matching of Attributed Relational Graphs", CAR-TR-376, CS-TR-2073, July 1988

ABSTRACT: This report presents experiments with a parallel algorithm for matching attributed relational graphs. The algorithm generates a state space tree in a breadth-first manner and then evaluates the tree by computing the edit distance for each candidate solution. The parallelization method used is best suited for MIMD-type computers. The first target machine is the Butterfly Parallel Processor, in which the programs were developed on Uniform System software supporting a shared memory model of computation. The second multiprocessor is a link-oriented Transputer-based system. In this system, concurrent processes communicate through message channels. The experiments show that nearly linear speedup can be achieved by parallelizing the algorithm in the outermost loop.

3. S. Ziavras and L.S. Davis, "Fast Addition on the Fat Pyramid and its Simulation on the Connection Machine", CAR-TR-383, CS-TR-2093, August 1988

ABSTRACT: This paper presents an algorithm for fast addition on the fat pyramid. The fat pyramid is a pyramid in which the storage space and the processing power allocated to a single node increase as the root of the pyramid is approached. The addition algorithm is based on a carry-lookahead technique. The computation time of the algorithm is proportional to $\log p + q$ for operands of size $p * q$ bits, when p processors are used to deal with the numbers. The addition algorithm was simulated on the Connection Machine; some performance results are presented in this paper.

4. T. Seppänen and K. Pehkonen, "A Generalized Algorithm for Border Tracking with an Implementation on the Butterfly Parallel Processor", CAR-TR-388, CS-TR-2099, August 1988

ABSTRACT: This report describes a generalized one-pass algorithm for border tracking of objects in thresholded binary images. The input image is scanned from top to bottom, from left to right. On each row, partial border descriptions produced on previous rows are updated according to run ends on the current row. Borders are represented by crack code strings following the outer borders in a clockwise direction, and the inner borders in a counterclockwise direction.

Secondly, a parallelized version of the algorithm is presented with an implementation on a Butterfly Parallel Processor. The program was developed based on the Uniform System approach which supports a shared memory model of computation. The input image is partitioned into equal sized blocks, and each partition is assigned to a separate processor. Partially completed border descriptions gathered from the blocks are finally merged in parallel.

5. E.V. Krishnamurthy and S.G. Ziavras, "Matrix g-Inversion on the Connection Machine", CAR-TR-399, CS-TR-2125, October 1988

ABSTRACT: The generalized inversion of a matrix has many applications. This report considers the implementation of the Ben-Israel-Greville algorithm for finding the Moore-Penrose inverse of a matrix. This algorithm is highly suitable for data-level parallelism and has several advantages: linearity, stability, reliability, determinism and scalability. Connection Machine experiments with random matrices of different dimensions are reported.

6. E.V. Krishnamurthy and S.G. Ziavras, "Complexity of Matrix Partitioning Schemes for g-Inversion on the Connection Machine", CAR-TR-400, CS-TR-2126, October 1988

ABSTRACT: Theoretical results concerning partitioning of large matrices for g-inversion are outlined. The complexity and performance analysis of these methods on the Connection Machine are described. It turns out that the use of the virtual processor configuration on the Connection Machine is of comparable efficiency to using any partitioning scheme, when the multiplicative iterative scheme is used for g-inversion.

7. E.V. Krishnamurthy and S.G. Ziavras, "Grid Evaluation-Interpolation on the Connection Machine using Tensor Products and g-Inversion", CAR-TR-401, CS-TR-2127, October 1988

ABSTRACT: Tensor products are widely used in the evaluation and interpolation of functions as well as 2D and 3D image blocks. This report describes the implementation of the tensor product method on the Connection Machine and its applications.

8. E.V. Krishnamurthy and S.G. Ziavras, "Multivariate Spline-Blending Approximation on the Connection Machine", CAR-TR-402, CS-TR-2132, October 1988

ABSTRACT: This report describes the principles of the projection operator technique for multivariable cardinal spline-blending approximation on the Connection Machine. This technique requires data-parallel operations for polynomial (single and multivariable) evaluation and hence is best suited for implementation on the Connection Machine. The basic operations needed are the inner product and the tensor product of vectors whose components are polynomials or their evaluated values. The spline-blending approximation has several applications: finite-element methods, digital image processing, optical flow and topography.

9. K. Kanatani and D. DeMenthon, "Reconstruction of a Road Shape from Images: A Computational Challenge", CAR-TR-413, CS-TR-2167, December 1988

ABSTRACT: A new scheme is presented for reconstructing the 3D shape of roads from camera images for the purpose of navigating autonomous land vehicles (ALVs). The formulation is based on the *local flatness approximation.* All equations are written in terms of it NHC vectors defined by quantities directly observable on the image plane. Hence, analysis is done solely in the image domain: No 3D solution is constructed in the scene. Much consideration is given to computational stability with regard to possible inaccuracy of image data. We propose a relaxation scheme which gradually guarantees the global consistency of the computed solution, and discuss its computational aspects. We also analyze the singularities of the constraint resulting from the local flatness approximation.

10. T. Bestul, "A General Technique for Creating SIMD Algorithms on Parallel Pointer-Based Quadtrees", CAR-TR-420, CS-TR-2181, January 1989

ABSTRACT: This paper presents a general technique for creating SIMD parallel algorithms on pointer-based quadtrees. It is useful for creating parallel quadtree algorithms that run in time proportional to the height of the quadtrees involved but that are independent of the number of objects (regions, points, segments, etc.) which the quadtrees represent. The technique makes use of a dynamic relationship between processors and the elements of the space and object domains being processed.

11. C.A. Sher and A. Rosenfeld, "A Pyramid Hough Transform on the Connection Machine", CAR-TR-421, CS-TR-2182, January 1989

ABSTRACT: A pyramid programming environment on the Connection Machine is presented. The mapping between the Connection machine and pyramid structures is based on a scheme called Shuffled 2D Gray Codes. A pyramid Hough transform, based on computing the distances between line or edge segments and enforcing merge and select strategies among them, is implemented using this programming environment.

12. K. Pehkonen, "The Implementation of the Linnainmaa and Harwood's Pose Determination Algorithm on Shared and Distributed Memory Parallel Computers", CAR-TR-423, CS-TR-2191, February 1989

ABSTRACT: This report describes the implementation of Linnainmaa and Harwood's pose determination algorithm on the Butterfly Parallel Processor (BPP) and Hathi 2 parallel computers. The architecture of the BPP is based on a shared memory model, whereas the Hathi 2 is based on a distributed memory model. The algorithm is computationally very intensive, which makes it suitable for parallel processing. The program is parallelizable using the processor farm technique, thus enabling automatic load balancing. The experiments show that the algorithm is very easy to parallelize. Furthermore, comparison of the two architectures show that the Hathi 2 is much more powerful than the BPP. Due to different implementation technologies, however, it is not possible to say whether one of the architectures is better than the other.

13. L.S. Davis and P.J. Narayanan, "Efficient Multiresolution Image Processing on Hypercube Connected SIMD Machines", CAR-TR-430, CS-TR-2227, April 1989

ABSTRACT: We describe two approaches to efficiently processing small images on hypercube connected SIMD machines. The first approach, called fat images, is based on distributing the bits representing the gray level (or other feature) from each pixel across the processors of a sub-hypercube, using Gray coding techniques to obtain a good mapping of the fat image into the hypercube. The second method, called replicated images, involves generating as many copies of the small image as will fit into the machine, and then distributing the computation of basic image processing operations across the copies. For the replicated images, we present algorithms for histogramming, table lookup and convolution, and describe the results of implementing the convolution algorithm on a 16K processor Connection Machine II.

14. S. Huang and L.S. Davis, "Parallel Iterative A* Search: An Admissible Distributed Heuristic Search Algorithm", CAR-TR-434, CS-TR-2233, April 1989

ABSTRACT: In this paper, a distributed heuristic search algorithm is presented. We prove that the algorithm is admissible and give an informal analysis of its load balancing, scalability, and speedup. A flow-shop scheduling problem has been implemented on a BBN Butterfly Multicomputer using up to 80 processors to empirically test this algorithm. From our experiments, this algorithm is capable of achieving almost linear speedup on a large number of processors with relatively small problem size.

15. E.V. Krishnamurthy, "What Can the Petri Net Model Offer to Neural Network Studies?", CAR-TR-446, CS-TR-2257, June 1989

ABSTRACT: This paper explores the possible application of a Petri net-like device as a node in neural network models. Such a model *called a Petri neural net* can be of great value in modelling neural processes exhibiting concurrency, asynchrony, intentionality and execution, nondeterministic choices (influenced by environment and learning, and independent of environment), inhibition, livelocks, deadlocks and divergence. This is illustrated by examples of Petri net modelling of multistable phenomena in vision. Specific analysis methods are

described to understand the behavior of Petri neural nets. It is shown how properties such as categorization and content addressability exhibited by neural nets respectively correspond to the problems of reachability and controllability in Petri nets. We also study some properties and limitations of existing neural nets and show how their computing power can be improved by including Petri net-like devices. The concept of Petri neural net can further be extended to include probabilistic concepts and this will lead to adaptive Petri neural nets which can be subjected to maximum entropy analysis. However, this analysis could become computationally intractable and even undecidable.

SUMMARY: Petri nets play a very important role in understanding the behavior of complex asynchronous concurrent and distributed computing systems. However the potentialities of the Petri net and related parallel computational models have not been explored in the study of neural networks and their modelling. We have shown how some of the salient aspects of Petri net and other models can possibly be combined with the existing neural network models to improve the modelling power of the latter and facilitate their simulation in concurrent asynchronous and parallel synchronous machines using concurrent programming languages such as Ada, CSP and Occam 2. We illustrated the use of the Petri net model for multistable phenomena in vision. We suggested using a Petri net-like device as a node in a neural net. We also discussed how Petri neural nets may be analyzed using linear algebraic techniques based on consistency checks. The use of consistency checks in Petri nets, as well as in Hopfield neural nets, was illustrated by examples. This study showed that Hopfield and other related neural nets essentially model proof by refutation in propositional logic by linear algebraic methods. In this sense they are no more powerful than a subclass of Petri nets called "marked graphs" and deterministic context-free production systems. We also dealt with stochastic Petri net-like devices and their learning behavior. In conclusion we indicated how Petri net-like devices can play an important role in the design of neural networks although their analysis could lead to computationally intractable and undecidable issues.

The examples on multistable phenomena in vision clearly demonstrate the potential of Petri neural nets. Presently, neural net models do not deal with asynchrony, concurrency, nondeterminism, intentionality and execution; Petri neural nets would permit inclusion of all these aspects thereby enlarging their scope to hybrid (digital and analog) neural nets. Also existing neural nets essentially model propositional logic via linear algebraic systems which are no more powerful than context-free production systems; unless the neural nets include facilities that can model predicate logic involving binding, unification, substitution and resolution their decision power cannot be increased. However this may result in computationally complex and undecidable issues. The Petri net model can be fitted together with the concept of tensor-product networks. This should enable us to understand coordination and motor learning behavior. Further research in this direction would permit the cross-fertilization of ideas from formal language theory, logic, computability and complexity theory, thereby improving our understanding of the immensely complex domain of neural nets.

16. E.V. Krishnamurthy, "Semantic Petri Nets—Applications to Multiple Inheritance and Knowledge Acquisition Systems", CAR-TR-447, CS-TR-2258, June 1989

ABSTRACT: This paper demonstrates the use of inhibitory Semantic Petri Nets (SPIN) for

a topological network-oriented approach to default reasoning in multiple inheritance systems having exceptions. The basic principles of this approach are explained and illustrated by examples. The superiority of SPIN over the Touretzky graph model is demonstrated.

SPIN can be extended to include probabilities, priorities, delays and time-outs. Hence it will be extremely useful for applications in semantic information based on connectionist model and intelligent decision support systems. Also its logical structure permits its implementation in concurrent programming languages such as Ada and Occam. We also briefly explain the relationship between SPIN and temporal logic. Also we indicate its applicability to the cover and differentiate and propose and revise heuristic classification used in knowledge acquisition, diagnostic, and other related expert systems.

SUMMARY: Inhibitory Petri nets can be used for parallel knowledge representation. We have found that they are superior to NETL, marker propagation machines, and the directed graph model of inheritance system described by Touretzky.

An inheritance system is a representation system founded on the hierarchical structuring of knowledge. In these inheritance systems, knowledge is organized by first creating abstractions. By abstraction, we mean a collection of properties shared by members of a set. For example, sheep and humans share the properties of being mammals. When abstraction is organized by inclusion relations such as the one described above we can represent this by a directed tree; this tree is known as a "taxonomic hierarchy" or "inheritance hierarchy", The taxonomic hierarchy provides an efficient method of representation and makes search very efficient in knowledge-based systems. When several properties are orthogonal or overlapping, we get the situation of multiple inheritance where we have an abstraction that holds for most members but with certain exceptions. In such complex situations the tree representation is usually inadequate and there is a need for the use of a more general directed graph. We have found that the use of inhibitory Petri nets provides a very clean semantics in this situation and permits efficient parallel representation of knowledge and its retrieval. This is due to the powerful property of inhibitory Petri nets which can represent first order predicate logic efficiently and also generate and recognize context-sensitive and type-0 languages. We have also briefly explored the relationship between SPIN and temporal logic.

Knowledge representation and the associated control strategy play a key role in the design of expert systems. In particular, matching task requirements with the knowledge representation and control strategy is the key issue in knowledge engineering. SPIN is a good step in this direction, since (1) it is based on a very fundamental model of concurrent computation that pervades concurrent computer architectural and language design along with the associated interwoven pattern of data and control flow, and (2) it works on the intention-action mode that can take into account the inhibitory, nondeterministic and probabilistic behaviors which are essential to intuition and creative reasoning in human expert systems.

17. R. Waltzman, "Geometric Problem Solving by Machine Visualization", CAR-TR-454, CS-TR-2291, July 1989

ABSTRACT: The goal of this research was to investigate automatic non-logical reasoning techniques based on principles of visualization in geometrical domains and to develop a methodology for implementing these techniques in computer programs. At the heart of the methodology that I developed is the idea of a representation formalism for geometric objects that is both isomorphic and canonical and completely integrates metrical and topological information. Next there is a defined set of problem-solving operations for storing, retrieving, and manipulating objects represented in this formalism. These operations are unified by the fact that they each entail some type of search or construction process that is highly constrained by the structure inherent in the formalism and, *"via isomorphism"* by the geometrical structure of the objects involved. I refer to these operations collectively as machine visualization techniques. Finally, there is a programmable problem-solving system in which various functional components are implemented using these visualization techniques. While the problem-solving system interpreter directs the actions of the system's functional components, the behavior of the interpreter itself is controlled by the programmer through problem-solving heuristics written in a pattern matching language that is a direct extension of the basic representation formalism. This language allows the programmer to express constraints on problem-solving activity directly in terms of the geometrical structure of the problem. Thus, both low level and high level problem-solving activities of the system are highly constrained by problem structure. As a concrete illustration of this methodology, I have implemented a geometric problem-solving system for solving a subclass of three dimensional packing problems and have successfully applied the system to solving two non-trivial three dimensional jigsaw puzzles (formulated as packing problems).

SUMMARY: Our methodology is based on a representation formalism for geometric objects that integrates metric and topologic information. We define problem-solving operations for storing, retrieving and manipulating objects. These operations are constrained by the structure of the formalism and by the geometric structure of the objects. We call these operations *machine visualization* techniques. The behavior of the interpreter which controls the operations can be controlled by the programmer through problem-solving heuristics written in a pattern-matching language. As a demonstration of the power of this approach we solved two non-trivial three dimensional jigsaw puzzles.

Human reasoning can be broadly divided into two categories: logical and non-logical. Until now, research in artificial intelligence has concentrated almost exclusively on logical reasoning. However, human problem-solvers use powerful reasoning techniques other than logic that are rooted in fundamental visualization capabilities. We constructed a model of such reasoning that consists of several components. The essence of this model is a formalism which expresses the geometric constraints of the problem in a way which makes them available to the set of problem-solving operations. The formalism applies specifically to convex or concave polyhedra and is isomorphic to the actual structure of the polyhedra, in the sense that there is a one-to-one mapping between the formal representation and the actual structure of the polyhedra. Consequently, the formalism totally integrates the metric and topologic informations of the objects. Another powerful feature of the representation is that it is intrinsic, i.e., independent of any frame of reference. Therefore the operations can manipulate objects without any consideration of their location or orientation in space.

More specifically, polyhedra are thought of as being composed of polygonal faces joined together by surface edges to form specific dihedral angles. Polygonal faces are circular ordered sequences of edges of specific lengths joined together at particular angles. The ordering of face edges, the signs of the angles at which they are connected, and the signs of the dihedral angles connecting the faces serve to define the orientation of the polyhedral surfaces. This results in a two-layered structural organization that is implemented as a planar graph whose nodes and arcs themselves have structured values. Polyhedra are also thought of as being composed of three dimensional vertices which are joined together by surface edges of specified lengths. Vertices are thought of as circular ordered sequences of face angles of specified sizes joined at particular dihedral angles. Analogous with the face-centered decomposition, the ordering of the face angles and the signs of the dihedral angles connecting them determine the orientation of the polyhedral surfaces. The face centered and the vertex centered decompositions of a given polyhedron are dual to each other in the sense that one can be unambiguously and automatically derived from the other. Therefore the same data structure can be used for both decompositions. This offers many advantages in terms of the problem-solving operations that can be defined.

In practical terms the expression *machine visualization* refers to the collection of operations used to store, retrieve and manipulate the representation described above. For example we discuss the importance of being able to "see" when two different pieces are identical in shape and size. The system does this by comparing the representation of the two polyhedra, exploiting the two layered structural composition of the representation. It takes one of the nodes of the face-centered graph representation of the first polyhedron and checks to see if it matches one of the nodes of the graph representation of the second polyhedron. If no match is found then the two polyhedra are different. However, if a match is found, the match acts as an *anchor point* from which to do the remaining comparisons. Thus comparing the remaining nodes is no longer a case of comparing circular lists. Each attempted match is a simple list comparison. The two graphs are then jointly traversed using a breadth first search with appropriate second layer comparisons (dihedral angles for arcs and faces for nodes) made at each step.

The method described above is effective when we want to compare only two polyhedra. However, to determine whether one polyhedron is identical to any number of other polyhedra, we introduce a method that uses an additional data structure called an *overlay graph*, that stores structural information regarding a set of polyhedra. The representation is unique. Common pieces of geometric structures share common corresponding parts of the storage data structure. Only local structure information is stored for recognition purposes. Overlay graphs are used to sort a set of polyhedra into groups of identical objects. We begin with an empty overlay graph. For each polyhedron in the set we first check to see if the structural information for that object is contained in the graph. If it is, then we take the name of the polyhedron associated with the structure and add the current object to the corresponding group. Otherwise we add the structural information for the current object to the overlay graph. The same technique is used to store old search states. We create an overlay graph that we use to store the structural information for the containers associated with each state. Then, for each new state we determine whether the containers associated with that state appeared in a previous state by checking that overlay graph.

Another important visualization technique is that of *adjoinment*, i.e., putting two pieces together to form a new piece. When two pieces are brought into contact, certain faces remain unaltered while others either disappear entirely because they are completely covered by the faces of the other polyhedron or are modified because they are only partially covered. Because the representation is isomorphic and intrinsic to the actual geometric structure, it is possible to implement adjoinment as a straightforward splicing operation between the representations of the objects being adjoined independently of location or orientation.

Another operation that is important to problem-solving is recognizing equivalent orientations of a given object. This operation uses a rotational symmetry detection algorithm. Piece placement instructions often have the form: Place piece A in container B so that vertex a of A coincides with vertex b of B. If vertex a' of A is equivalent to a under a rotational symmetry, then looks exactly the same from the viewpoint of a as it does from a'. This means that placing piece A in container B so that a' is coincident with b will result in placing A in an orientation and location equivalent to that resulting from the placement specified in the original instruction. The two-layered structure of the representation admits the use of a generate and test algorithm where both the generate and test parts of the algorithm are simple searches that are highly constrained by the structure of the representation.

Tentative piece placement decisions are heuristically made on the basis of structural features of the pieces to be placed, as well as the container into which they are to be placed. However the final decision to place a piece depends on whether it actually fits into the desired location. To answer these question we have introduced a specialized containment algorithm that is based on a spatial localization process. The result of this process is to distribute the surface boundaries of the polyhedra in question into volume sectors so that surface boundary comparisons (checks for intersection) can be done simply and need be done only within each sector. Once containment is established, the piece is placed by subtracting its volume from that of the container. This subtraction process is similar to the adjoinment operation in that it is only a question of splicing together the representations of the piece and the container.

The system performed very well on two jigsaw puzzle examples. The first puzzle had six pieces. The possible number of state computations required to solve the puzzle without making any use of the problem structure is probably more than 27,000. The system was able to solve this puzzle in roughly 10 state computations. The second puzzle was much harder and had 12 pieces. A naive approach would require more than $3 \times 10^{2}9$ state computations. The system was able to solve the puzzle in roughly 200 state computations.

This work is perhaps the first demonstration of a problem-solving system that solves non-trivial problems using isomorphic representations and non-logic based reasoning techniques. The pattern matching language for expressing problem-solving heuristics makes the system general and applicable to a broad class of problems. More importantly, it concretely illustrates the potential of a new methodology for constructing reasoning systems.

18. E.V. Krishnamurthy, "Modelling Dynamically Constrained Distributed Programs", CAR-TR-455, CS-TR-2292, July 1989

ABSTRACT: Semantic Petri Net (SPIN) modelling of dynamically constrained distributed programs is discussed. Unlike Leler's constraint graph and Numao's cell-relation model, SPIN can be automated for consistency checking based on T-invariants during the update propagation. Hence it is useful for the design of robust distributed programs in declarative languages.

SPIN can also model data-structure-resident parallel processes having different granularities.

Further studies on temporal logic based Petri nets are needed to understand dynamically constrained distributed systems.

SUMMARY: Constraint programming is a new paradigm in which the programmer states a set of relations (including exceptions and constraints) among a set of well-defined objects. The system then finds a solution that satisfies these relations together with exceptions and constraints. The constraint programming system has applications in areas such as CAD, CAM, graphics and AI, to mention only a few. Leler defines a general purpose specification language based on a constraint graph model, that allows a user to describe a constrained satisfaction system using rules. However, this approach does not provide a systematic reasoning procedure for answer extraction and consistency checking. Also the constraint graph is not a suitable model when the objects themselves are processes defined on data types among which relations are specified, e.g., concurrent programs arising in robotics, service and protocol specification in asynchronous distributed systems. We have shown how a Semantic Petri Net (SPIN) can model constraints, as well as data-structure-resident processes. Reformulating a constraint program as SPIN rather than as a constraint graph has several advantages: (1) The constraint graph model is a passive network; hence, it cannot express intentions and actions. SPIN, however, provides the intention-action mode that can be converted to guarded Horn clauses for constraint satisfaction. (2) The constraint satisfaction during an update propagation in a Petri net corresponds to the firing of the different transitions; this amounts to computing a non-zero T-invariant for the incidence matrix of the Petri net. This is equivalent to using resolution to prove that a set of clauses contains a contradiction, hence it is an algorithmically superior model in comparison to the constraint graph of Leler for the representation as well as analysis of constraint programs. (3) SPIN can model distributed programs that include data-structure-resident processes, such as queues, arrays, etc.

19. A. Basu and J.(Y.) Aloimonos, "An Efficient Algorithm for Motion Planning of Multiple Moving Robots", CAR-TR-457, CS-TR-2297, August 1989

ABSTRACT: The problem of coordinating the motion of multiple robots of the same size translating in the plane is examined. First the complexity of the decision problem is shown to be polynomial. Then the complexity of the problem of obtaining the final configuration from the initial one with the minimum number of moves is determined to be NP-complete. Finally, we present an efficient algorithm for solving the problem with expected time $O(n^2\sqrt{n})$, provided the distance between the initial and final configurations is at most of the same order. This complexity can be further improved since the proposed algorithm is parallelizable.

SUMMARY: We have addressed the problem of efficiently coordinating the motion of multiple objects in a discretized environment when there is sufficient space to guarantee that a solution to the problem exists. Here, as in several previous papers, we considered a special case of the problem. In particular we considered a rectangular workspace area and all elements as squares. This subcase of the problem has applications in assembly of electronic circuits where the components are laid out on a grid, memory management in distributed systems, mobile robots in a structured workspace.

Consider the problem of 2D memory management. The programs can be thought of as rectangles in a 2D workspace (memory). One may encounter the problem of rearranging the programs so as to collect the free memory space from time to time (this problem is also known as compaction). The problem would then be to devise efficient algorithms to perform this task.

Another problem of interest is rearrangement of goods in an automated warehouse. It is easy to design simple robots which are capable of pushing the boxes (containing goods) around. Goods may be used up over time in a warehouse and replacement usually occurs in bulk. To help in efficiently retrieving items one may be interested in rearranging the warehouse at certain times depending on changing demand patterns. If multiple robots capable of pushing boxes exist, then one would be interested in efficient parallel algorithms for warehouse rearrangement.

In some factory environments robots can be made to follow certain paths on a grid. Usually the grids are laid out below the factory floor and each robot is programmed to pick up some specific signals. The problem here is how to efficiently coordinate the movement of many robots in a cluttered environment.

The difficulty in the problem is inherently due to the shortage of free space. The algorithm works as follows. In the first step the original rearrangement problem is divided into smaller problems. These subproblems are solved locally, eliminating the need for priority graphs. Elements are then exchanged across the boundaries. The process of solving subproblems locally and exchanging across boundaries continues until some intermediate final configuration is obtained. Space is rearranged from the intermediate final configuration to obtain the final arrangement. The crucial step in the process is how to divide the original problem into subproblems. For this an iterative scheme and a recursive scheme are described. The recursive procedure is shown to have a better average case performance. We have implemented a distributed and parallel version of our algorithm.

20. B. Kamgar-Parsi, P.J. Narayanan and L.S. Davis, "Reconstruction (of Rough Terrain) in Range Image Shadows", CAR-TR-458, CS-TR-2298, August 1989

ABSTRACT: The common approach to estimating the surface elevation at a given point is based on weighted averaging of the elevations of the neighboring points. This approach often yields incorrect results for surface reconstruction in range image shadows; the elevation of the shadow region is often overestimated such that it could not have been occluded in the first place. In this paper we propose an algorithm that does not yield such incorrect results, and preserves the statistical properties of the surface. The method is best suited for reconstruction of random stationary surfaces, such as rough terrain. It has applications in off-road autonomous land navigation.

21. T. Seppänen, "Speed-up Analysis of Parallel Programs with an Image Analysis Program as a Case Study", CAR-TR-459, CS-TR-2302, August 1989

ABSTRACT: In this report we preliminarily introduce a method for efficiency analysis of parallel systems. The method is constructed on the concept of critical paths of concurrent programs. Utilizing timing profiles of a program running on the desired target computer, a model explaining the speed-up behavior of the program is constructed. The model will be later used for pointing out those factors that seem to affect the speed-up behavior of the program. An image analysis algorithm is implemented on the Butterfly Parallel Processor (BPP) and the experiment is used as a testbed for the method. The image analysis program performs a gray-level connected components analysis and feature extraction for area-segmented images. A gray-level connected components analysis is first carried out to generate an area-segmented labelled image. Then intra-area and inter-area features are computed for the area segments, including adjacency graphs. Parallel computation is achieved by dividing the input image in equal sized blocks and assigning each block to a different processor of the BPP, according to the principles of data parallelism. An asynchronous slave process is attached to each image block, and an asynchronous master process controls the slaves via synchronizing barrier variables. Mutual exclusion among the slaves is implemented with locking primitives. Both of the main stages of the program contain three steps: first, image blocks are processed locally as long as possible; second, a description of block interfaces is created to be utilized in the third step, in which the partial results are merged.

SUMMARY: The good match of our model to the timing measurements indicates that the essential features of our concurrent program influencing its speedup behavior have been grasped. The program described above includes several parallelizable loops and hence provides a good vehicle for the development of analysis methods. We demonstrate that if a concurrent program can be divided into consecutive, separately parallelizable segments, which are synchronized to each other by appropriate techniques, the speedup analysis of the resulting concurrent system can be carried out by analyzing the segments separately with the proposed critical path analysis approach, and then combining the subresults to achieve the properties of the whole system.

22. B. Kamgar-Parsi and B. Kamgar-Parsi,"On Problem Solving with Hopfield Neural Networks", CAR-TR-462, CS-TR-2310, August 1989

ABSTRACT: Hopfield and Tank have shown that neural networks can be used to solve certain computationally hard problems; in particular, they studied the Traveling Salesman Problem (TSP). Based on network simulation results they conclude that analog VLSI neural nets can be promising in solving these problems. Recently, Wilson and Pawley presented the results of their simulations which contradict the original results and cast doubts on the usefulness of neural nets. In this paper we give the results of our simulations that clarify some of the discrepancies. We also investigate the scaling of TSP solutions found by neural nets as the size of the problem increases. Further, we consider the neural net solution of the Clustering Problem, also a computationally hard problem, and discuss the types of problems that appear to be well suited for a neural net approach.

SUMMARY: In a seminal paper, Hopfield and Tank showed that neural networks can be used to solve computationally hard problems such as the Traveling Salesman Problem (TSP). To investigate how well the network performs they simulated its behavior and found very encouraging results, in that the network frequently finds valid solutions of high quality. Recently, Wilson and Pawley reported the results of their simulations of the Hopfield and Tank solution of the TSP. Their results differ from those presented by Hopfield and Tank in two respects: (i) the number of trials yielding valid solutions is considerably less; and (ii) the solutions found by the network are not much better than randomly selected tours.

We have performed simulations of the Hopfield and Tank solution of the TSP. We found that, indeed, the number of times the network succeeds in finding valid solutions is considerably less than that found by Hopfield and Tank. However, we found that when the neural net finds a valid solution it is of remarkably good quality. Further, we showed how the success rate of the network in finding valid solutions can be markedly improved.

We also investigated how the neural net solution of the TSP scales with the size of the problem. The results are not encouraging, in that the scaling is poor. Although the quality of solutions that are found by the network remain good, finding valid solutions becomes increasingly difficult as the size of the problem increases. This suggests that neural nets may not be suitable for solving computationally hard problems. However, there does not appear to be a universal answer to this question, because there are other computationally hard problems, such as clustering, that appear to be well suited for the neural network approach and have good scaling properties.

In deciding whether the Hopfield model of neural networks is suitable for solving a computationally hard problem, one has to consider two factors: (i) how good are the solutions; and (ii) what is the success rate for finding valid solutions.

It appears that when the analog network finds a solution, its quality is generally very good. A specific solution, of course, depends on the chosen initial state. There is overwhelming empirical evidence (based on Monte Carlo estimates) that deeper minima of the energy function of an analog network have generally larger basins of attraction; thus a randomly selected initial state has a higher chance of falling inside the basin of a deep minimum and finding a very good solution. Hopfield neural nets are complicated dynamical systems, and as yet, there are no theoretical estimates for the sizes of these basins. Therefore, one cannot give the probability of finding the best solution. However, one can claim that analog networks greatly enhance the probability of finding very good solutions. This is true for both problems—TSP and clustering—we have investigated in this work.

The success rate appears to be dependent on the problem. For TSP the success rate is rather modest; in particular, it scales poorly as the size of the problem increases, which suggests that neural nets may not be suitable for solving TSP. For the clustering problem, on the other hand, the success rate is very high, nearly 100%, and its scaling as the number of points increases is excellent. The main difference between these two problems is in their matrix representations of valid solutions. A given tour in the N-city TSP problem is represented by a $N \times N$ permutation matrix with constraints on both the rows and the columns, which is a hard syntax to satisfy. A given partitioning of N points among K clusters in the clustering problem is represented by a $K \times N$ matrix with constraints only on the columns, which is a much easier syntax to satisfy. This appears to be the reason for the much higher success rate of the clustering problem in finding valid solutions. Furthermore, the scaling of the success rate with problem size may be related to the density of "on" neurons in the syntax matrix, i.e., the ratio of 1's to the total number of elements in the syntax matrix. For TSP this ratio is $N/N^2 = 1/N$, which scales inversely with the size of the problem N. For the clustering problem this ratio is $N/KN = 1/K$, which scales inversely with the number of clusters K. Since in most problems K is small and much less than N, increasing the number of points should not adversely affect the scaling, which is confirmed by our results.

Based on these observations, it appears that problems that are suitable for a neural net approach are those whose valid solutions can be represented with few constraints, i.e., easy syntax. From simulations, we find that neural nets converge to solutions very rapidly, within a few τ. The simulations, however, require a great deal of computer time as the network must be updated thousands of times. The real benefit of neural nets, however, may lie in the future when they can be mapped on analog chips. Analog VLSI neural nets are being developed; these devices have very fast processing times in the micro to millisecond range, thus making their use in solving suitably chosen computationally hard problems very attractive.

23. S. Chandran, "Merging in Parallel Computational Geometry", CAR-TR-468, CS-TR-2333, October 1989

ABSTRACT: In this report we consider problems in parallel computation and computational geometry. The goal is to come up with a solution that is as fast as possible by using a large number of processors, without being too inefficient in the total number of operations performed by the processors. For a problem of size n we will typically employ n processors, and will accept $O(\log n)$ time algorithms as fast. If we perform about the same number of operations as the sequential algorithm, the algorithm will be deemed cost-optimal.

We provide a cost-optimal parallel solution to the problem of computing the area of a union of rectangles that runs in logarithmic time. The previously best known parallel algorithm runs in $O(n)$ time, and the parallel lower bound for this problem using n processors is $O(\log n)$. We are able to do this by constructing an efficient parallel data structure using the parallel *plane-sweep* technique and pipelined merge sort.

The same technique can be used to solve other problems in VLSI and computer vision cost-optimally. Problems in dynamic computational geometry, for instance, demand a variation on the above scheme. These are rather straightforward to tackle in the uniprocessor version: we introduce the *dynamic merging* technique to solve these problems in the parallel context. The algorithm is cost-optimal and runs fast. Sometimes parallel solutions can be obtained by finding new geometric insights rather than developing new algorithmic techniques. We are able to unify the solution to the problem of computing inscribed and circumscribed approximations to a large sided convex polygon by one such characterization. Previous algorithms for the two problems were different and had more complex correctness proofs. We also substantially improve the time complexity to $O(\log\log n)$ using $\log\log n$ processors. This is one of the first few examples of a non-trivial parallel algorithm in computational geometry that runs in sublogarithmic time.

An orthogonal issue is that of characterizing parallel algorithms. Theories for sequential computing developed over the years cannot be readily adopted for parallel computation. Cost-optimality is one indication of a good parallel algorithm. We suggest that the number of processors times the square of the time (i.e., pt^2) is another indication. The analogy to the quantity "area-time-squared product" (i.e., at^2) in VLSI is obvious. Our models essentially emphasize to the user of parallel machines that different parallel algorithms may need to be employed depending upon the problem size and multiprocessor size.

24. S. Kambhampati, "Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach", CAR-TR-469, CS-TR-2334, October 1989

ABSTRACT: Generating plans from scratch is a computationally expensive process. A general framework for minimal modification and reuse of existing plans in new problem situations can significantly improve the efficiency of this process. Although some past planning systems addressed this problem, their methods did not pay adequate attention to the flexibility and minimality of modification.

This report provides an integrated framework for flexible reuse and modification in hierarchical nonlinear planning. The framework, implemented in the PRIAR system, is based on the *validation structure* of the plans. The validation structure is a formal representation of the internal dependencies of the plan. It is annotated on the plan by the planner to make later reuse more effective.

Reuse of a plan is formally characterized as the process of repairing the inconsistencies that arise in its validation structure when it is used in a new problem situation. The repair process is domain-independent and exploits the validation structure to remove any inapplicable parts of the plan and to suggest additional high level refit tasks to be achieved. The planner is then invoked to reduce these refit tasks, and is guided in this process by a heuristic control strategy which utilizes the validation structure of the plan. The heuristic strategy localizes the refitting by minimizing the disturbance to the applicable parts of the plan. The validation structure is also used to judge the utility of reusing the plan in a given new problem situation, forming the basis for plan retrieval.

The flexibility and the efficiency of plan modification in the PRIAR framework are evaluated both through empirical studies and through formal characterization of the reuse processes. The coverage of its modification techniques for hierarchical nonlinear planning is formally characterized in terms of the plan validation structure.

25. D. DeMenthon and L.S. Davis, "New Exact and Approximate Solutions of the Three-Point Perspective Problem", CAR-TR-471, CS-TR-2351, November 1989

ABSTRACT: Model-based pose estimation techniques which match image and model triangles require large numbers of matching operations in real world applications. We show that by using approximations to perspective, lookup tables can be built for each of the triangles of the models. Weak perspective approximations have been applied previously to this problem; we consider two other perspective approximations, paraperspective and orthoperspective. We obtain analytical expressions which are as simple as those obtained by weak perspective. These approximations have much lower errors for off-center images than weak perspective. The errors of these approximations are evaluated by comparison with exact solutions. The error estimates show the relative combinations of image and triangle characteristics which are likely to generate the largest errors. The corresponding cells of the lookup tables can be flagged, so that object pose calculations would disregard image and model triangle pair combinations when their characteristics correspond to the flagged cells.

26. Z. Chen, "A Flexible Parallel Architecture for Relaxation Labeling Algorithms", CAR-TR-474, CS-TR-2355, November 1989

ABSTRACT: The design of a flexible parallel architecture for both the discrete relaxation

labeling (DRL) algorithm and the probabilistic relaxation labeling (PRL) algorithm is addressed. Through proper space-time arrangement of the computation steps involved, the relaxation labeling processes can be run on a systolic-array like architecture in linear time for each iteration. Thus a high degree of computation parallelism is obtained. The arrays use one-dimensional, one-way communication lines between adjacent PEs and interface with the external environment through only a single I/O port. Because of the hardware simplicity and programmability features of the PEs, the architecture is well suited for VLSI implementation and is flexible enough to execute different relaxation algorithms. An illustrative example of running a region color labeling problem on the proposed architecture is described and a general running procedure is also given. Finally, performance comparison between the proposed architecture and other existing ones is presented in some detail.

SUMMARY: Relaxation labeling algorithms have been applied successfully in a number of fields, for instance, signal restoration and identification analysis, language identification, graph or subgraph homomorphism, image segmentation and pattern recognition, scene interpretation and understanding problems. One of the major issues of the labeling algorithm is that it is computationally intensive and typically requires exponential time with a single-processor architecture. Many researchers have presented multiprocessor approaches and tried to build fast architectures to support relaxation labeling algorithms because the problem can be divided into mutually exclusive subproblems and is well suited for parallel processing. These techniques are often faced with difficulties in task partitioning, scheduling and synchronization for relaxation operations among multiprocessors. Or they may have an appealing run time performance from a theoretical viewpoint, but suffer from physical realization problems such as data communication congestion and I/O routing complexities. Most of these approaches remain in the phase of virtual software. Lately, there is an increasing interest in implementing relaxation operations by using dedicated VLSI hardware architecture that can be built in a modular fashion, for instance, in the form of systolic arrays. Several preferential factors of the systolic approach, such as the feasibility of pipelining, high degree of parallelism, and avoidance of global communication, make the arrays suitable for VLSI fabrication. The underlying idea behind these methods is the application of suitable transformations to the relaxation algorithms to obtain a representation that can be easily mapped onto their proposed architecture. Thus, different relaxation labeling algorithms give rise to different array designs. However, the function of the resultant systolic array is somewhat restricted simply because the applied architecture is too fixed to cover different applications. In our work, we considered the relaxation labeling algorithms in two different forms: discrete relaxation labeling (DRL) and probabilistic relaxation labeling (PRL). Both of them are solvable on our proposed systolic architecture. The arrays use one-dimensional and one-way communication lines between adjacent PEs and interface with the external environment through a single I/O port. Because of the hardware simplicity and programmability features of the PEs, the architecture is well suited for VLSI implementation and is flexible enough to be adaptable to a number of applications, without compromising both speed and hardware complexity. Moreover, the proposed architecture runs in linear time for each iteration of the labeling process, and is also able to detect the consistency status of DRL as well as the convergence condition of PRL at the hardware level without host involvement. On the other hand, the use of one-way flow between the arrays simplifies

the circuit design and the system can be converted to self-timed arrays to avoid the clock skew problem for large labeling processes.

We have introduced a flexible parallel architecture for relaxation labeling algorithms. A transformation scheme is used for parallel computation of the supporting evidence so that a high degree of parallelism is feasible. We simplify the control structure of the relaxation operations to clear away the difficulty of complicated data communication between objects and their neighbors. We preload the compatibility coefficients which will reside in each PE so that only the object data need to be circulated among the PEs. This speeds up the process. The one-dimensional and one-way layout of systolic arrays is suitable for fault tolerant and self-timed circuit design; thus it increases reliability and avoids the clock skew problem. Our performance evaluations show that the proposed architecture is generally superior to existing systems.

In order to verify the design of the proposed systolic array and the combiner module, a simulation software package called the DAISY system has been used to check the specification. We have finished the logic simulation to check the timing sequence for the architecture operation and the alphanumeric simulation of signals to verify the intermediate processing results. The experiments show that the proposed architecture is working properly. Future research includes the development of the VLSI custom chip and proper modification for covering a wide spectrum of related algorithms.

27. S. Huang and L.S. Davis, "Speed-Quality Tradeoffs in Heuristic Search", CAR-TR-486, CS-TR-2396, February 1990

ABSTRACT: In this paper, we study speed-quality tradeoffs in heuristic search both theoretically and empirically. We start by analyzing a heuristic search algorithm which is equivalent to the weighted heuristic search algorithm originally suggested by Pohl. We show that this algorithm, when using a heuristic with "constant relative error", can find an optimal solution with *linear* complexity, as opposed to A^* which has *exponential* complexity. We then prove some theorems to illustrate that the performance of this algorithm is determined by the *relative* accuracy of heuristics which may in some cases *improve* with increasing distance from the goal, in contrast to that of A^* which is determined by the *absolute* accuracy of heuristics which almost always *deteriorate* with increasing distance from the goal. New dynamic weighting schemes which are to some extent opposite to what Pohl had suggested are proposed. Finally, a new heuristic search algorithm which attempts to exploit speed-quality tradeoffs is presented. This algorithm was tested on the Flow-Shop Scheduling Problem and detailed experimental results are provided.

SUMMARY: We present a new heuristic search algorithm which exploits speed-quality tradeoffs. It describes the notion of "constant relative error" of a heuristic function, in which the ratio of the difference of the estimated error and the actual error to the estimated error is constant for all nodes. We show that a particular existing heuristic search algorithm can produce an optimal solution in linear time provided that the heuristic function used has constant relative error. We prove various theorems relating the relative accuracy of heuristic functions to the performance of the heuristic search algorithms. The new heuristic search algorithm presented has the property that it computes ever-improving solutions as it progresses to the final optimal solution. Thus, it could be used in a speed-quality tradeoff scheme

in which sub-optimal solutions could be provided in situations where time was critical, and better solutions could be provided where time constraints were more relaxed but solution quality was important. The new algorithm has the flavor of simulated annealing, in that it begins by quickly finding a solution of unknown quality, and then incrementally improves the solution as computation progresses. However, unlike simulated annealing which relies on random processes to improve the solution quality, the new algorithm relies on the heuristic function itself. The algorithm is based on an evaluation function which is a weighted sum of the cost found to reach a node and the estimated cost remaining to a goal node, i.e., the heuristic function. However, through successive iterations improving the solution, the values of the weighting factors are adjusted based on the cost of the best solution found so far. With each iteration the adjustment of weighting factors tends to favor the cost-to-node summand over the estimated-remaining-cost summand in the evaluation function. This process is shown to eventually produce the optimal solution to the search problem. Detailed results of experiments with the flow-shop scheduling problem are given.

28. S. Huang and L.S. Davis, "Sequential and Parallel Heuristic Search under Space Constraints", CAR-TR-497, CS-TR-2438, March 1990

ABSTRACT: Practical applications of heuristic search algorithms, such as A^*, are often thwarted by memory limitations. The memory limitation of A^* is overcome by the Iterative-Deepening-A^* (IDA^*) algorithm. It has been claimed and is widely believed that IDA^* is asymptotic time optimal. We argue in this paper that this claim is only true for some limited problems such as the 15-Puzzle but is not true generally. A sequential and a parallel heuristic search algorithm which not only overcomes memory limitations, but also makes effective use of available memory to better exploit space-time tradeoffs are then presented. These algorithms have been tested on the Flow-Shop Scheduling Problem and experimental results are provided.

SUMMARY: We analyze an existing, linear-space variation on A*, called IDA*, which is widely believed to be asymptotic time optimal, and presented evidence which contradicts this belief. Through careful argumentation we demonstrate that the asymptotic time optimality of IDA* depends on the existence of a negative correlation between the number of "ties" of values of the evaluation function, taken over the set of state-space nodes, and the magnitudes of the node values themselves. We argue that this correlation holds for the particular search problems and heuristics discussed by the proponents of IDA* but does not hold for some other interesting search problems and heuristic functions. The existence of this correlation was embodied in a particular assumption of the proponents of IDA*, specifically that the algorithm searches deeper in the search tree with each new iteration. Besides this analytical evidence, we describe experimental evidence of the non asymptotic time optimality of IDA* obtained from experiments with the flow-shop scheduling algorithm.

We then describe and analyze a new tree search algorithm, AD*, which is a hybrid of the A* and IDA* algorithms, and which operates by alternating between two modes: the A* mode is used until a preset limit M on the number of nodes on the OPEN list is reached, and the IDA*-like mode is used where this limit is exceeded. The difference between the IDA*-like mode of AD* and IDA* itself is that with each iteration, after updating its threshold value, IDA* performs a depth-first search from the start node of the search tree, whereas the

IDA*-like mode of AD* performs a depth-first search from one of the nodes on the OPEN list. By changing the value of M appropriately, the algorithm can provide a desired space-time tradeoff. We analyze results of experiments performed using AD*. We also analyze a parallel version of AD*, called PAD*, which was based on our own earlier PIA* algorithm, and we obtain experimental results for PAD*.

29. L.T. Chen and L.S. Davis, "A Parallel Algorithm for List Ranking Image Curves in $O(\log N)$ Time", CAR-TR-501, CS-TR-2458, April 1990

ABSTRACT: This paper describes an algorithm for ranking the pixels on a curve in $O(\log N)$ time using a CRCW PRAM model. The algorithm accomplishes this with N^2 processors for an $N \times N$ image. After applying such an algorithm to an image, we are able to move the pixels from a curve into processors having consecutive addresses. This is important on hypercube-connected machines like the Connection Machine because we can subsequently apply many algorithms to the curve (such as piecewise linear approximation algorithms) using powerful segmented scan operations (i.e., parallel prefix operations). The algorithm was implemented on the Connection Machine, and various performance tests were conducted.

30. K. Pehkonen, D. Harwood and L.S. Davis, "Parallel Calculation of 3D Pose of a Known Object in a Single View", CAR-TR-502, CS-TR-2459, April 1990

ABSTRACT: This paper describes a selective generate-and-test algorithm for SIMD-parallel calculation of the 3D pose of a known object from a single perspective view. The algorithm consists of three main stages: object pose estimation, optimization, and reliability analysis. The first stage involves parallel generate-and-test of candidate pose solutions obtained by selectively matching model and image point triples, testing their correspondence by parallel transformation of all visible model points and comparison of their features with the image. Instead of exact algebraic calculations, the three point perspective pose estimation problem is solved numerically. In the second stage, the initial pose estimate is optimized using a least-squares method. In the final stage, the reliability of the optimized pose is estimated using covariance analysis. The simulations showed that the approximate initial pose estimates are sufficiently good to obtain very accurate optimized results. Furthermore, the processing times for an object with 12 vertices were on the order of a few hundreds of milliseconds on a Connection Machine-2 with 512 floating point units.

31. O. Silvén, "Estimating the Pose and Motion of a Known Object for Real-Time Robotic Tracking", CAR-TR-503, CS-TR-2461, April 1990

ABSTRACT: This report deals with the problem of estimating the pose and motion of a known object in three dimensions by using an initial estimate and a sequence of monocular images. The camera is assumed to be attached to a robot arm used for vision-controlled tracking of the object. The finite image resolution, robot errors, and changes of object motion during the non-zero delays for image acquisition and analysis introduce uncertainties into the measurements and estimation process. The uncertainties are handled by an extended Kalman filtering technique that produces object pose and motion estimates from feature point measurements in the images. The important problem of selecting the feature points used for estimation is also considered. Usually several feature points are available from an image frame, but because of various real-world constraints, using only some of them is justified. We present four methods for selecting a given number of feature points and compare them experimentally. With the test object that was used, random selection gave good results, and when the actual computational costs of the methods are taken into account, this approach becomes very attractive.

32. E. Puppo and L.S. Davis, "Surface Fitting of Range Images using a Directional Approach", CAR-TR-504, CS-TR-2477, May 1990

ABSTRACT: Range images can be represented as piecewise-smooth $2\frac{1}{2}$D surfaces. The segmentation of a range image requires low-level processing which filters the noise and estimates the surface normal at every pixel. Robust algorithms are needed which are able to perform the above tasks while preserving the image structure (surface discontinuities). Algorithms based on robust statistics often suffer from high computational complexity and low efficiency in the presence of high-variance noise. In the paper we propose a new method for processing images of polyhedra. The method is based on the application of a one-dimensional algorithm to slices of the image taken along several directions. Results obtained through the one-dimensional processing are subsequently integrated to produce two-dimensional results. The algorithm exhibits low complexity, high efficiency and high parallelism. Moreover, the method can be adapted to any kind of noise by modular substitution of the basic one-dimensional algorithm.

33. L.T. Chen and L.S. Davis, "Parallel Algorithms for Testing if a Point is Inside a Closed Curve", CAR-TR-511, CS-TR-2513, July 1990

ABSTRACT: This paper describes two parallel algorithms to test whether a point is inside a digitized closed curve. The first algorithm assumes the curve is stored in a linear array of processors. The second algorithm assumes the digital image containing the curve is stored in a 2D array of processors and that a clockwise (or counterclockwise) trace of the curve is available (i.e., each point on the curve has a pointer to the next clockwise point). Both algorithms assume that scan instructions are available on the processor array. The algorithms take constant numbers of scan instructions.

34. T. Seppänen and V. Vuohtoniemi, "Speedup Analysis of a Program for the Generalized Hough Transform on Two Different Multiprocessors", CAR-TR-517, CS-TR-2551, October 1990

ABSTRACT: A theory of performance analysis of parallel systems was introduced in an earlier report. Here it is demonstrated how the theory works in practice by applying it to a parallel program for a generalized Hough transform (GHT). Two parallel computers were used in the experiments, a Butterfly Parallel Processor (BPP) at the University of Maryland and a HATHI multiprocessor at Abo Akademi, Finland. The architectures of these computers differ in their communication network: the BPP has a distributed shared memory for data exchange, whereas in the HATHI, the processors communicate by sending messages via serial links.

We analyze timing profiles of parallel execution, or lists of timing measurements achieved in each active processor of a parallel system executing the selected application program. The modeling of a parallel program starts by the segmentation of the program into virtual actions at the selected level of abstraction and then proceeds by aggregating the virtual actions to construct the descriptions of higher levels of abstraction until the desired level is achieved. The execution time of each virtual action is modeled to be a function of the system size or the number of active processors. In this way a separate analysis can be applied to each virtual action at a given level of abstraction to compare the performance differences of the computers or two slightly different versions of the same program in one computer. The hierarchical structure of the model enables fast location of those program segments which act as bottlenecks for performance.

After our successful experiments, we believe that the speedup analyzer introduced here is a useful tool in the final steps of parallel system development in which the execution performance is enhanced and the system size (number of processors) is minimized.

35. S. Arya, D. DeMenthon, P. Meer and L.S. Davis, "Textural Analysis of Range Images", CAR-TR-518, CS TR-2552, October 1990

ABSTRACT: This paper addresses the problem of texture discrimination in range images. The range data is transformed to a common coordinate system, and then resampled to generate a regular grid of points in order to eliminate the effect of different sampling rates. Ten statistical textural features based on co-occurrence matrices are computed on the resampled data, and are used to discriminate between two classes of natural surfaces consisting of pebbles of different sizes lying on a plane. Seven of the ten textural features are found to be useful in discriminating between these two classes. The experiments also confirm the importance of resampling the data before computing the textural features.

36. Z. Pizlo and A. Rosenfeld, "Recognition of Planar Shapes from Perspective Images Using Contour-Based Invariants", CAR-TR-528, CS-TR-2577, December 1990

ABSTRACT: This paper presents a new method of recognizing a two-dimensional shape from a single perspective projection image taken from an unknown (three-dimensional) viewpoint. The method is based on quadratic approximations to the effect of the slant of an inverse perspective transformation on angles and lengths. These approximations allow us to define contour-based properties that are invariant under perspective transformation. The method can be used to recognize partially occluded shapes, as well as shapes that are not exactly related by perspective transformations. When a shape is recognized, the method also provides estimates of its tilt and slant.

37. J.(Y.) Aloimonos and A. Rosenfeld, "Visual Recovery", CAR-TR-531, CS-TR-2580, January 1991

ABSTRACT: A large part of computer vision research treats image understanding as a recovery problem, i.e., as the problem of estimating properties of the scene from images. This was viewed, during the 1980s, as a necessary first step before attempting to perform visual tasks. We describe this approach from a unified perspective, emphasizing recovery of shape (i.e., of scene geometry). We also discuss the limitations of the approach, possible directions for future research, and possible alternative approaches.

38. Y.A. Teng, D. DeMenthon and L.S. Davis, "Stealth Terrain Navigation", CAR-TR-532, CS-TR-2586, January 1991

ABSTRACT: In this paper, we propose a framework for solving visibility-based terrain path planning problems on massively parallel hypercube machines. A typical example is to find a path that is hidden from moving adversaries. This kind of problem can be generalized as a time-varying constrained path planning problem, and is proven to be computationally hard. The framework we propose is an approximation based on both temporal and spatial sampling. Since a 2D grid-cell representation of terrain can be embedded into a hypercube with extra links for fast communication, our framework can be very efficient when implemented on hypercube machines. The time complexity is in general $O(T \times E \times \log N)$ using $O(N)$ processors, where T is the number of temporal samples, E is the number of adversary agents, and N is the number of grid cells on the terrain. We also show that our framework can be applied to several realistic problems with a variety of path optimizations. All algorithms have been implemented on the Connection Machine CM-2 and results of experiments are presented.

39. P.J. Narayanan and L.S. Davis, "Replicated Data Algorithms in Image Processing", CAR-TR-536, CS-TR-2614, February 1991

ABSTRACT: Data parallel processing on processor array architectures has gained considerable popularity in data intensive applications such as image processing and scientific computing as massively parallel processor array machines became feasible commercially. The data parallel paradigm of assigning one processing element to each data element results in an inefficient utilization of a large processor array when a relatively small data structure is processed on it. Popular algorithmic techniques such as the divide-and-conquer technique reduce problems involving large data structures to those involving smaller ones. The large degree of parallelism of a massively parallel processor array machine does not result in a faster solution to a problem involving relatively small data structures than the modest degree of parallelism of a machine that is just as large as the data structure. In this paper, we present an algorithmic technique, called "replicated data algorithms", that uses more processors than the number of data elements and achieves considerable improvement in performance over conventional data parallel algorithms both analytically and in practice. The technique combines the traditional paradigms of data parallelism and operation parallelism using multiple copies of the data structure. In this paper, we demonstrate the technique for two image processing operations, namely, image histogram computation and image convolution, and present the results of implementing them on the Connection Machine. In each case, we also compare the replicated data algorithm with the conventional data parallel algorithm on three common interconnection networks, namely, the 2D mesh, the 3D mesh and the hypercube, to determine the conditions under which the technique of data replication speeds up the computation. Finally, we demonstrate the generality of the data replication technique in image processing by showing how replicated data algorithms can be developed automatically for any operation that can be described as an image to template operation using image algebra.

40. J. Ohya, L.S. Davis and D. DeMenthon, "Recognizing 3D Objects from Range Images and Finding Their Six Pose Parameters", CAR-TR-538, CS-TR-2616, February 1991

ABSTRACT: A method for recognizing polyhedral objects from range images and for finding their six pose parameters is studied. The approach consists of building an object library; pre-processing; hypothesis generation for the six pose parameters; and hypothesis verification. In the object library, information necessary for hypothesis generation and verification is stored for each object. The pre-processing steps calculate image features at each pixel in a range image. In hypothesis generation, image features are chosen at random from range images and matched to object model features. The rotation and translation required for the match are computed to yield data points in parameter spaces. The LMS (Least Median of Squares) method, a robust clustering method, generates hypotheses for object pose from the data points. The generated pose is verified by a similarity measure which evaluates how well a model with the generated pose would fit the original range image. Hypothesis generation and verification are performed for all models in the library, and object recognition and pose estimation discussions are based on choosing the most similar result. Experiments using synthetic range images have provided insights into the problems that can arise with this approach.

41. P. David, D. Harwood and L. Davis, "A Probabilistic, Local Feature, Stereo Correspondence Algorithm", CAR-TR-540, CS-TR-2628, March 1991

ABSTRACT: Stereo vision is a technique where, by matching points in two images of the same scene, the depths to these points may be computed. The hardest part of this process is usually considered to be that of determining the point correspondences. This paper describes a technique for computing stereo correspondences of the edges in two images. The goodness of a particular correspondence of two edge points is based on the probability that a set of local properties of each edge point was generated by imaging the same scene point. This probability incorporates information about the different types of edges that may be found in an image. Edge correspondences for the two images are determined by optimizing the goodness over the entire image, subject to left-to-right and edge-continuity constraints. We use a dynamic programming technique developed by Ohta and Kanade to perform this optimization. The algorithm is tested on a variety of images, and its performance is compared to that of several other algorithms.

42. L.T. Chen and C.P. Kruskal, "EREW Parallel Algorithm for List Ranking Image Curves in $O(\log N)$ Time", CAR-TR-541, CS-TR-2629, March 1991

ABSTRACT: This paper describes an algorithm for ranking the pixels on an image curve in $O(\log N)$ time using an EREW PRAM model. The algorithm accomplishes this with N processors for a $\sqrt{N} \times \sqrt{N}$ image, although it could be done optimally with $N \log N$ processors if we are willing to use a more complicated list ranking algorithm such as that of Anderson and Miller. Several complications that arise due to the restriction to exclusive read are discussed. Problems that arise when implementing the algorithm on a distributed memory parallel machine instead of a shared memory PRAM are also discussed. The algorithm was implemented on the Connection Machine, and various performance tests were conducted.

43. P. Burlina, D. DeMenthon, and L.S. Davis, "Navigation with Uncertainty: I. Reaching a Goal in a High Collision Risk Region", CAR-TR-565, CS-TR-2717, June 1991

ABSTRACT: We present a probabilistic method for noisy sensor based robotic navigation in dynamic environments. The method generates an optimal trajectory by considering as optimality criteria the probability of not colliding with the obstacles and the probability of accessing an operational position with respect to a moving target object.

Estimates of the obstacle's kinematic parameters and measures of confidence in these estimates are used to produce the probability of collision associated with any robot displacement. The probability of collision is derived in two steps: a stochastic model is defined in the kinematic state space of the obstacles, and collision events are given simple geometric characterizations in this state space.

44. P. Burlina, D. DeMenthon, and L.S. Davis, "Navigation with Uncertainty: II. Avoiding High Collision Risk Regions, CAR-TR-568, CS-TR-2724, July 1991

ABSTRACT: We present a probabilistic method for sensor based robotic navigation in dynamic and noisy environments. The method generates a trajectory that guarantees a tolerable associated collision risk. Estimates of the obstacle's kinematic parameters and measures of confidence in these estimates are used to produce regions where the probability of encountering any obstacle is bounded by a predefined value.

45. P.J. Narayanan and L.S. Davis, "Rank Order Filtering on Processor Array Machines", CAR-TR-587, CS-TR-2770, October 1991

ABSTRACT: Rank order filters form an important class of low level image operations that have widespread applications in image smoothing, texture analysis, etc. Processor array machines are popular in practical parallel processing, particularly in data intensive areas such as image processing. In this paper, we study several ways of computing rank order filters on processor array architectures, in the traditional data parallel framework of the machines. We also develop a replicated data algorithm for efficient computation of rank order filters of small images on relatively large processor arrays. Theoretical analyses of all algorithms are presented in the paper. We have implemented these algorithms on a Connection Machine and a MasPar MP-1. The implementation results are presented for all algorithms. We also make comparisons between theoretical and experimental results, wherever relevant.