The Journal of Supercomputing, 12, 37–56 (1998)

A Dual Source, Parallel Architecture for Computer Vision

A.M. WALLACE andy@cee.hw.ac.uk
G.J. MICHAELSON greg@cee.hw.ac.uk
N. SCAIFE norman@cee.hw.ac.uk
W.J. AUSTIN billa@icbl.hw.ac.uk
Department of Computing and Electrical Engineering, Heriot–Watt University, Riccarton, Edinburgh EH14 4AS

Editor: Hamid R. Arabnia

Abstract. We present a parallel architecture for object recognition and location based on concurrent processing of depth and intensity image data. Parallel algorithms for curvature computation and segmentation of depth data into planar or curved surface patches, and edge detection and segmentation of intensity data into extended linear features, are described. Using this feature data in comparison with a CAD model, objects can be located in either depth or intensity images by a parallel pose clustering algorithm.

The architecture is based on cooperating stages for low/intermediate level processing and for high level matching. Here, we discuss the use of individual components for depth and intensity data, and their realisation and integration within each parallel stage. We then present an analysis of the performance of each component, and of the system as a whole, demonstrating good parallel execution from raw image data to final pose.

Keywords: parallel vision; multi-source data; cooperative processing

1. Introduction

Vision is a complex process. In general, lower level processing operations such as edge detection or the computation of local curvature in either intensity or depth images require extensive processing due to the large image data structures employed, dealing typically with image sizes ranging from 256 by 256 to 640 by 480 pixels. Higher level processes such as feature-to-feature matching between image and model data are complex because of the number of permutations which must be evaluated, generally exponential in the number of scene or model features. Parallel algorithm development offers a potential solution to the problem of execution of these algorithms in *real* or *reasonable* time.

We have been developing parallel algorithms for all levels of vision, and have recently combined these to form a complete and concurrent parallel visual architecture which processes depth and intensity data at the same time. In particular, we have investigated algorithmic parallelism and redistribution of resources between algorithms combined with feature or model data parallelism at the high level, rather than simple spatial data parallelism at the low level. We have examined the behaviour of the separate processes and system as a whole as the operating parameters and data changed. Implementation has been in occam on a Meiko Computing Surface CS-1, a general purpose MIMD architecture based on T800 transputers [15, 9].

This paper describes the development and implementation of our current parallel architecture for cooperative depth and intensity based object recognition. The following sections

provide context for our work, describe the overall parallel architecture and individual components, and discuss the performance of the final system.

2. Related work

Although there has been considerable interest in the development of parallel algorithms for visual processes [15, 4, 19, 16], the bulk of this attention has been devoted to single algorithms in isolation rather than complete systems. Furthermore, the relative ease of data parallelism at the lower levels of the processing chain (e.g. edge detection, curvature computation) has dominated the published literature. Integration of several parallel algorithms into a complete system is a more challenging task, yet there have been some notable examples.

The Image Understanding Architecture (IUA) [18] integrated parallel processing within a visual hierarchy operating at three levels. Low level processing included the simple, SIMD operators such as edge detection for which easy data parallelism could be applied. Intermediate processing included grouping or segmentation processes, performing data reduction to establish scene description in terms of semantic objects. Higher level processing included interpretive matching, the formation and verification of object and scene hypotheses. This hierarchy is common, and indeed similar to the one employed here. However the major limitation of the IUA is the requirement for a specialised architecture with a different type of parallel processor at each level. This makes it difficult to generalize, replicate or disseminate the results of the project, although the processing times were understandably impressive, less than a second for individual algorithms. Concurrent integration of these algorithms is suggested, but not presented in the original publication.

Wang and Brady [17] discussed a heterogeneous machine, PARADOX, which, like our system, used T800 transputers, but in concert with a pipeline processor. Unlike our Meiko Computing Surface, but like the IUA, it was configured especially for a particular visual application, determining structure from motion in image sequences. In stark contrast to the work presented here, there is no potential for flexibility in the control architecture; the intention is to pipe an image through a pre-determined sequence of operations so that real time navigation of a vehicle can be achieved. Thus, it is complementary to our work.

A notable example of an integrated parallel vision system, and certainly the one which has most relevance, is the MARVIN system developed at the University of Sheffield [10, 11]. Like PARADOX, it uses dedicated pipelined hardware for low level convolutional processing at frame rates and a specially designed parallel configuration based on T800 transputers. Parallel C was used to program the algorithms within a custom-developed run-time environment. The processing path has similarities to the intensity route presented here, except for the use of stereo correspondence to obtain 3D scene features rather than inverse perspective transformation of 2D features in our case, and interpretation tree search rather than pose clustering to match scene and model data. The presented data on timing is actually contradictory, but both of the above articles refer to total processing times of the order of 10 seconds which is not dissimilar from those we have achieved for single route processing on the general architecture.

In summary, a key difference between our own and previous work is in our use of a standard, homogeneous parallel architecture. Specialised architectures may well offer

performance advantages but are seldom cost-effective or generally available. Our intention is to show that, in principle, a general purpose MIMD processor can be used for processes at *all* levels of the visual hierarchy, that good speed-up can be maintained, and that these processes can be integrated successfully. While actual processing times may be greater on homogeneous general purpose architectures, since programmed flexibility is unlikely to match dedicated hardware in any one application, such systems are much more widely applicable.

Another major difference lies in our dual source architecture; previous parallel systems used a single mode, usually intensity data, but we process depth and intensity data concurrently. Resource allocation is not fixed, but can be allocated to processors according to the known processing load of each algorithm.

3. System overview

Our original design supported cooperation between components specialised for particular visual data sources. We have concentrated on intensity and depth data though in principle our system would support components for other data sources. We wanted to provide a high degree of cooperation and control across and between stages. At each level, intensity and depth data might be processed independently or at the same time. Metrics from each level would then determine how resources should best be allocated at the next level. Thus, the system might decide to continue with both sources independently, to drop one source or to focus on combined processing. Furthermore, previous stages might be revisited to attempt to improve processing.

However, we had a prior commitment to implementing this architecture in occam on a Meiko Computing Surface, an MIMD system with 32 T800 transputers, each with only 4 Mbyte of memory. While this does not equate to the development of special purpose hardware discussed in the previous section, it does constrain our ability to realise the full flexibility of our design. Our architecture needs to retain intermediate data structures to enable prior processing to be revisited and, for large images, this places a considerable premium on memory. Regrettably, occam lacks dynamic memory allocation and all space has to be fixed at compile time, leading to the allocation of maximal rather than optimal amounts of memory. These considerations led us to a closely coupled process architecture in which the first two stages are combined so that intermediate structures may be shared. Thus, our final implementation has two stages which we will henceforth refer to as *segmentation* and *clustering*. These stages are independent but the realised architecture provides for a high degree of cooperative control between them. This is illustrated in figures 1 and 2. Both stages are based on process farms where a coordinating processor (farmer) allocates tasks to a "pool" or "farm" of worker processors. In general, the processors were configured as a grid with toroidal wrap-around, though with relatively few processors this was not critical to overall performance.

Each worker in the pool for segmentation may perform either intensity or depth processing. In the former case this consists of edge detection by the standard Canny operator [3] followed by an edge tracking process to form extended contours. In the latter case, the mean and Gaussian (HK) curvatures are computed locally and used to segment the image [2]. Typed

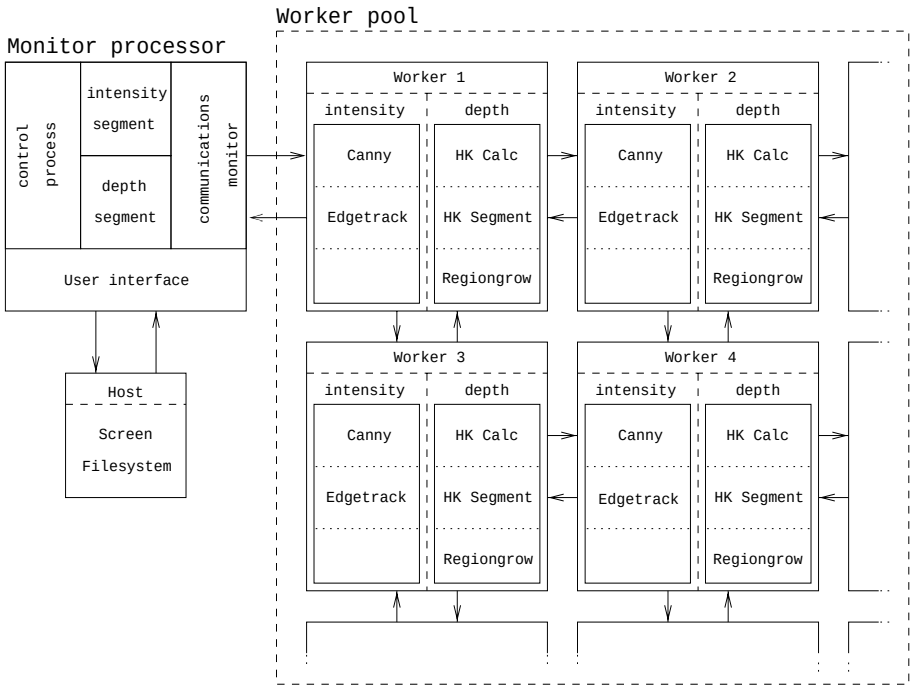


Figure 1. Parallel architecture for vision: segmentation

planar and cylindrical surfaces are defined by a subsequent region growing and surface fitting algorithm.

Each worker in the farm for pose clustering can perform object location in four phases, defined as leader clustering, cluster centre calculation, refinement and extraction of the best cluster data. Identification is possible if more than one model is used. Only the first phase of the clustering differs between intensity and depth data because perspective transformation is required in the former case. The other phases are identical.

Registered depth and intensity images form the input to our system. These can be generated synthetically by a modified ray tracing package or acquired by a triangulation rangefinder with an additional intensity camera described in [5]. Registration of data is performed using the known transformations between the three cameras, calibrated by a standard method [14].

To illustrate the processing involved, an example of a pair of depth and intensity images is shown in figure 3. On the left are shown, from top to bottom, the intensity data (shown from one camera viewpoint), the edge map derived from the Canny operator, and a subset of the straight tracked lines which are extracted from this edge map. On the right are shown the depth data, a visual representation of the HK curvature classified according to

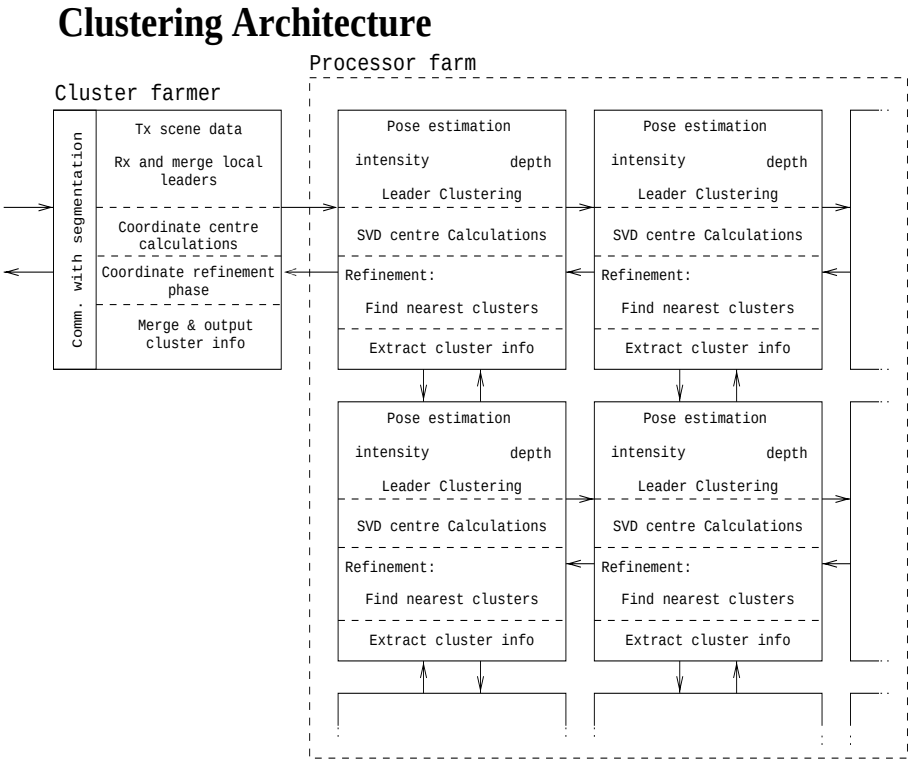


Figure 2. Parallel architecture for vision: clustering

sign (plus, zero, minus), and the surface patches which are grown from that data. The final image shows the CAD model superimposed on the intensity data in a position defined by the clustering of matched model and scene (intensity and depth) features.

4. Segmentation

4.1. Canny edge detector

The Canny operator [3], or its variants, are the most commonly used edge detection methods, consisting of a series of convolution operations followed by edge thinning through non-maximal suppression. A hysteresis thresholding process may then be applied. As implemented here, the first convolution is with a simple 1D Gaussian filter but is carried out in the x and y dimensions simultaneously. The smoothed images are then further convolved with a 1D derivative of a Gaussian filter in each direction orthogonal to the smoothing operation. This gives edge strength components in each direction which are resolved to give an edge magnitude and direction. The final stage is another local window operation whereby the pixel at the centre of a nine-pixel neighbourhood is compared with adjacent

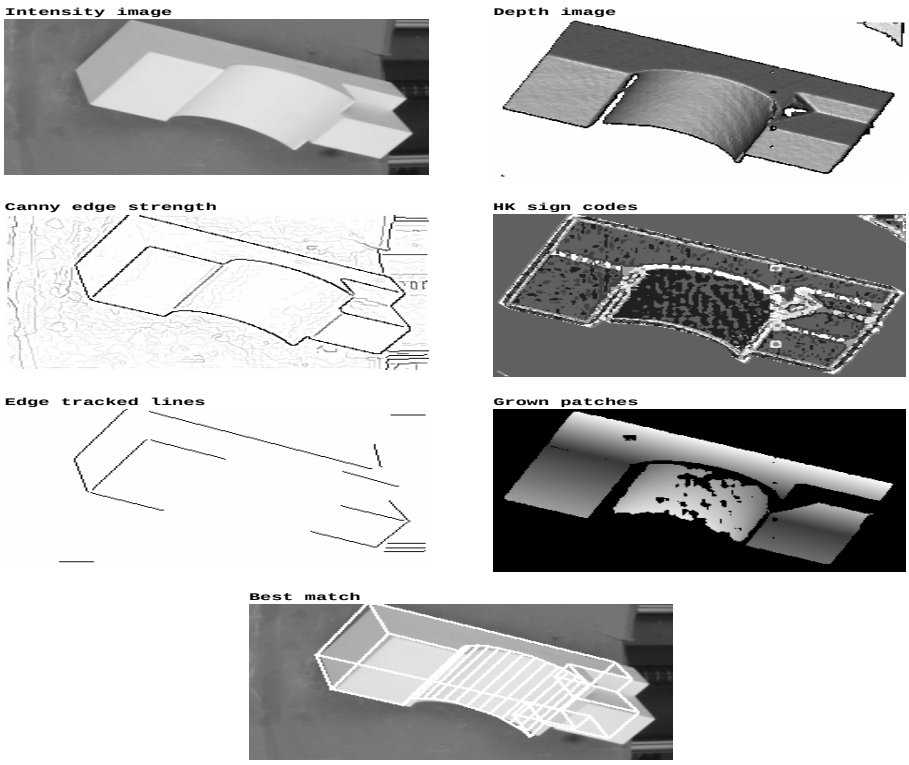


Figure 3. Parallel system processing for a simple block image

pixels normal to the edge directional component at the centre pixel and is suppressed if it has an edge magnitude lower than either of its neighbours.

Canny edge detection can be implemented as a simple and regular geometric decomposition of the image into rectangular sub-images, although some overlap is necessary to allow neighbourhood comparisons at sub-image boundaries. In this instance, the number of sub-images is exactly equivalent to the number of processors. This leads to speed-up characteristics which are just sub-linear, as illustrated by the performance data shown in figure 4 for up to 25 processors, using the data shown in figure 3.

4.2. Edge tracking

In our context, edge tracking is the linking of edge data together to form extended line segments. If these segments are of known and simple parametric form, such as the straight lines shown in figure 3, then Hough transformation can be used. This is relatively easy to make parallel by regular decomposition of either the image or parameter space; several references are given by Leavers [8]. If the shape of the line segments is irregular and not

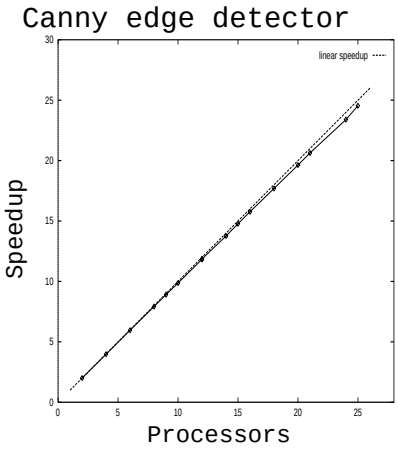


Figure 4. Canny phase parallel performance

known, in figure 6 for example, then it is more appropriate to use a tracker based on local edge continuity.

Parallel implementation of local edge tracking is perceived as a difficult problem [19] because communication patterns are not necessarily regular and are highly data-dependent. Thus, for example, simple geometric data composition is not possible because a contour may pursue a path anywhere in the image. In our solution, all points over a given strength threshold (1) are treated as seed points and a simple angular threshold (2) is used to guide the growing process. As parallel edge tracking of this form is not well-established we include a short algorithmic description below.

```
I = {edges in image}, LL = ∅, V = ∅
for each point p ∈ I, p ∉ V
    if edge-strength(p) ≥ threshold1
        set V = V ∪ {p}, L = {p}, G = {p}
        while G ≠ ∅
            g ∈ G, G = G - {g}
            for each point n in the neighbourhood of g, n ∉ V
                if compatibility-constraint(g,n) ≤ threshold2
                    set G = G ∪ {n}, L = L ∪ {n}, V = V ∪ {n}
        LL = LL ∪ {L}
```

`I` is the input image (edge strength and orientation), `LL` is the growing list of tracked lines, `V` is the set of points that have been added to lines, `L` is the line currently being grown and `G` is the set of compatible points in the neighbourhood of `L` still to be checked.

We have implemented several parallel options of the edge tracking module. Our best solution uses a pipeline of two generalised data farms, in which a farmer controls and allocates tasks to a set of worker processors, then collates the results as returned from the

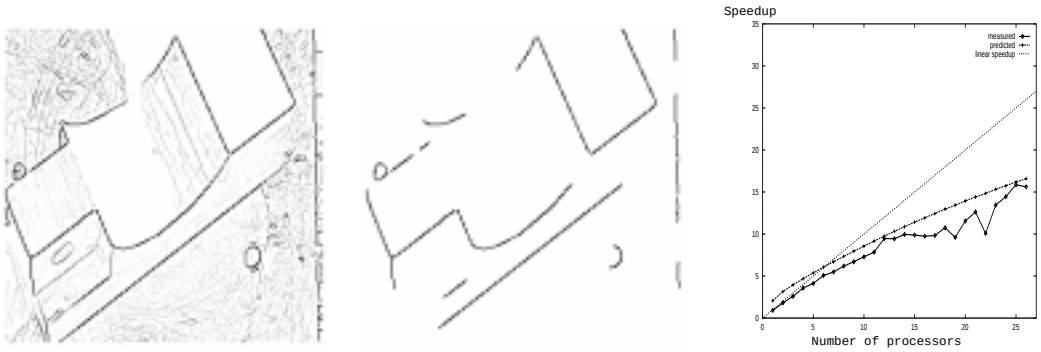


Figure 5. Tracking phase parallel performance

workers. The first parallel process performs tracking within sub-images, starting from seed points to form the initial line list, LL. Farming is preferable to geometric data decomposition as the edge data is generally sparse and irregular. The second parallel process refines LL by merging partial lines by comparing their end coordinates. This process has order $O(l^2)$ where l is the number of partial lines arising from the first phase, but can be pruned if necessary by removal of short spurs.

Figure 5 shows an example of the predicted and measured parallel performance for the data-farmed edge tracking algorithm for up to 25 processors on a simple block image similar to figure 3. In the complete parallel system, the tracked lines may be further fitted by either a straight line or elliptical function and rejected if a poor fit is obtained, since our model representation is derived from a CAD system [13] which does not allow higher order fits. These are not implemented in parallel, and impose a small sequential process overhead on the system as a whole. In our examples, figure 5 shows the fitted straight line and elliptical segments, but figure 6 shows another example of the algorithm applied to a facial image. In this case, the central picture shows the thresholded Canny data, and the right image shows tracked contours which have further strength and angular continuity thresholds applied, as shown in the algorithmic description. This illustrates the greater flexibility compared to Hough transformation, our parallel implementation of which is described elsewhere [9].

4.3. Mean and Gaussian curvature calculation

Computation of the mean and Gaussian curvature on the depth image is a common initial process for depth data processing [2] and can be implemented by a simple geometric decomposition of the image data in a similar manner to the Canny operator. The first and second order surface gradients in the x and y directions are calculated by convolving with sets of normalised polynomials. These gradients are then combined to give the mean and Gaussian curvatures which are then thresholded to give a sign value (negative, zero or positive) for each curvature. These two sign values can then be combined into a single sign code for each pixel. Since all these operations are separable the HK calculation can be

Figure 6. Edge tracking of a face image

carried out independently at each pixel. Figure 7 (left) shows the speedup achieved on the data of figure 3. Like the Canny operator it is almost linear.

4.4. Coarse HK curvature based segmentation

The sign map of mean and Gaussian curvature values is processed using a connected components algorithm. Each point in the list of current points is checked and neighbouring points which have the same HK sign value are added to the list. This continues until all points are considered.

For parallel implementation, geometric decomposition with interworker communication is used. Within each decomposed area, lists of points having common HK parameters are grown by adding adjacent points to initial seeds, analagous to the edge tracking module. Like the edge tracking module, boundary processing is more complicated. On reaching a boundary point, each processor has to check firstly, which processor is on the other side of the boundary and secondly that a message has not already been received from the processor on the other side of the boundary indicating that it has already reached that point. Lists of regions with common parameters must then be compared and merged to form global lists if they have adjacency and common parameters. When merging occurs, each processor maintains the local identity of the patch, but the monitor processor maintains a global list of which patches on which processors correspond; this is updated on termination of the worker processing. Then the data is distributed among the workers and there follows a tidy up operation where the master renumbers the patches for output to the next process.

Figure 7 (middle) shows the parallel performance achieved using this method which is considerably less than linear in this case. This is due to the inefficiency associated with geometric decomposition in this context. Not all data points represent patches which leads to an imbalance of data between processing elements in a geometric decomposition. This could be cured by the provision of a mask plane which defines processor/data point allocations. Before the main algorithm commences, the actual data in the image could be divided equally between workers, defined by this mask plane, which would be transmitted prior to running the main algorithm. However, this has not been implemented due to

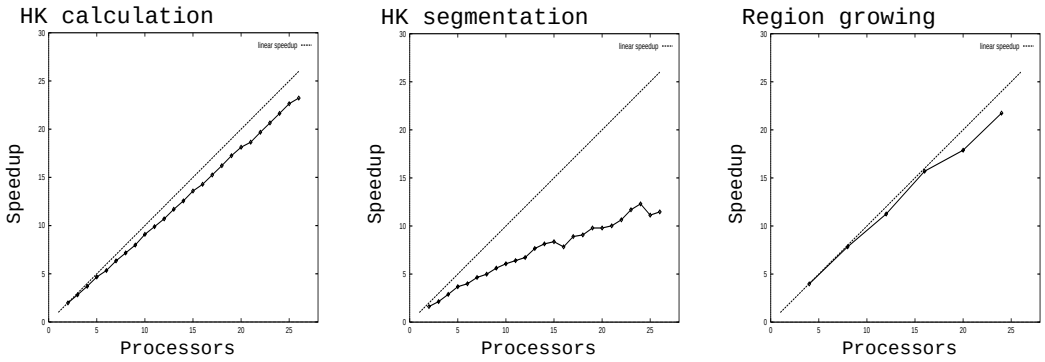


Figure 7. Speedup for HK calculation, segmentation and region growing

limitations imposed by the static memory allocation as even within the single domain the local memory is fully utilised.

4.5. Region growing from the coarse segmentation

Once a coarse segmentation based on the signs of mean and Gaussian curvature has been achieved, the next step is to sample the patch and perform region growing by least square fitting of parametric surfaces. Least square fitting is sensitive to outliers, but repeated sampling and selection is used to circumvent this [7]. The growing process is similar to the connected components algorithm above, but in this case neighbouring depth points are chosen if their residuals from the current fit are less than a global threshold.

Region seeding is implemented in parallel using data farming, where the total number of samples required for each patch is divided up into chunks to give control over the granularity. Region growing is implemented in parallel using task farming. Here, the initial seed regions are treated as tasks and a fine-grain geometric decomposition of the image is used to define task boundaries. Region growing is performed within these subimages and when growing crosses a geometric boundary a list of external seed points is maintained. Upon completion within the subimage, these points are partitioned according to their subimage number and new tasks are formed from the combination of these points plus their subimage number which are returned to the monitor for queueing. Again there is an optimal granularity for this process, too fine and communications dominates, too coarse and there is not enough parallelism. Figure 7 (right) shows the speedup achieved for a suitable level of granularity of figure 3. In this case, the image is divided up into 30×30 rectangular subimages. Although unlike the simple geometric decomposition of the Canny and HK parameter computation, the data farming approach has nevertheless resulted in near-linear speedup, primarily because the pre-segmentation of pixels required for the fitting process is complete and boundary checking is not required.

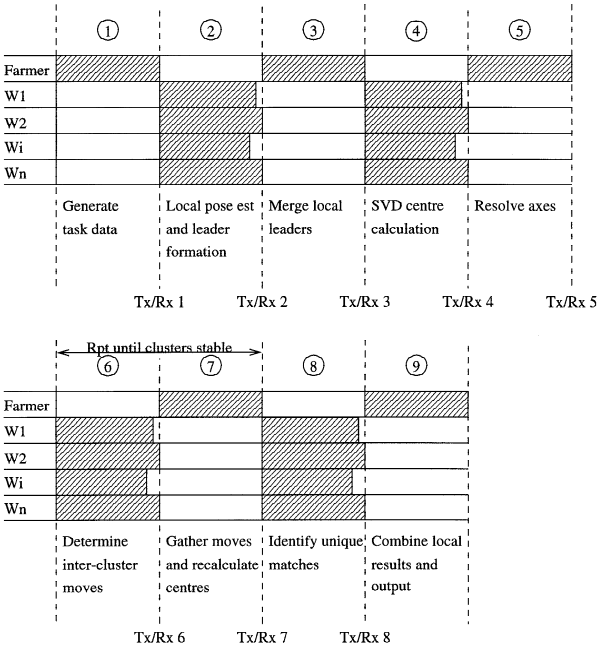
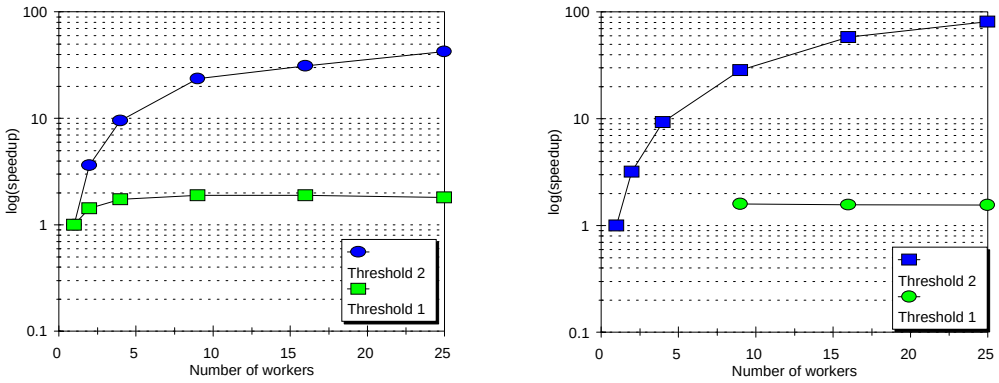


Figure 8. The main processes and messages in the parallel pose clustering algorithm.

4.6. Skeleton based implementation

The segmentation stage was implemented using algorithmic skeletons [6]. A skeleton is a generalised code template which is formed by abstracting away from functional details in a control structure. Subsequently, a skeleton may be instantiated with problem specific functional details. Skeletons are particularly appropriate for parallel programming where the same communication and control constructs tend to recur.

Four related divide and conquer skeletons were composed to form this stage: geometric decomposition (GD), data farming (DF), task farming (TF), and geometric decomposition with interworker communication (GDIW). Each skeleton maps a function over an image structure through a farmer parcelling out sub-image structures to workers for parallel processing. The skeletons differ in how the image structure is partitioned and in the amount of worker-worker and worker-farmer communication. GD is used when the image is divided into equal sub-images with equal processing requirements. DF is more appropriate for irregular data. TF generalises DF to enable the result from each worker to be sent out again by the farmer for further processing. GDIW generalises GD to enable communication between workers during processing, for example to resolve the boundary conflicts. Table 1 summarises the deployment of skeletons to each component, showing the code re-use and correspondences between the two routes, e.g. the use of simple GD for edge detection and



(a) Depth-based speedup
Serial time 3.00s
25 worker time 0.0710s

(b) Intensity-based speedup
Serial time 62.1s
25 worker time 0.767s

Figure 9. Speedup for the real data shown in figure 3

HK calculation. Each skeleton is realised within the combined process farm architecture for the segmentation stage shown in figure 1.

5. Pose estimation

To locate, and if necessary identify simple objects, we use a parallel pose clustering approach [1, 12]. A candidate pose is computed from each minimal subset of matched model and scene features, in this case the planar and curved patches derived from the depth data and the contours (and contour intersections) derived from the intensity data. Each matched surface or contour represents an entry in the 6D pose space, for 3 translational and 3 rotational parameters. A leader clustering algorithm is employed which makes no a-priori assumptions about the number or distribution of these data entries. Large clusters indicate probable models and poses, and the centroid of the cluster provides an estimate of the transformation matrix.

There are four basic phases in the clustering algorithm:

1. Generation of pose estimates from matched model and scene features:
   (A) using depth data;
   (B) using intensity data.
2. Cluster assignment and determination of parameters
3. Cluster refinement and inter-cluster movement of features
4. Identification of unique matches between features within clusters

Table 1. Summary of skeleton deployment

Module	Skeleton
Canny edge detector	GD
Edge tracking	DF → DF
HK calculation	GD
HK segmentation	GDIW
Region growing	DF → TF

Phase 1 depends on the different orthographic or perspective transformations used during image formation, and on the different segmentation algorithms used to extract surface and boundary features. However, since a common 3D point-vector type [13] can be used to represent both 3D surfaces and 3D space curves, phases 2, 3 and 4 are common to depth and intensity data. Thus, the object pose may be derived from a single data source, or from both together. In this instance, the toroidal grid offers a good compromise between communication costs and extensibility. It allows relatively short distances from the farmer to the furthest worker, but can be extended by the simple addition of a further row or column of processors.

The parallel algorithm is refined into 9 stages shown in figure 8; data communication between farmer and workers are shown as Tx/Rx1 to Tx/Rx9. Parallel advantage is gained from the worker processing, that is stages 2, 4, 6 and 8 which correspond to the four stages in figure 2. At stage 2 pose estimates are computed in parallel from pairs of scene and model features; this differs between intensity and depth data because of the perspective transformation in the former case. As pose estimates are computed they are assigned to existing leader clusters, or form new leaders according to a compatibility threshold. Calculation of cluster centres can also be done in parallel. Cluster refinement is the heart of any clustering process; in general this is iterative but in practice we have found that the leader clustering process works well, and few iterations are required. When refinement is complete the workers identify the unique model-scene matches that contributed to the clusters, and these are combined by the farmer to define the most probable pose or poses.

Figure 9 shows one example of the speedup characteristics that are obtained when clustering is applied to the segmented depth and intensity data shown in figure 3. The data are plotted for up to 25 processors, but an additional thresholding parameter of "cluster size" can be seen to have a significant effect on the performance. This is due to the splitting of clusters between workers in the farm when running the parallel algorithm. If a threshold of one is applied (i.e. singleton patterns may define a cluster) then the parallel algorithm will do much more work than the sequential equivalent since cluster merging will take place both on the farmer and the workers. Consequently, the speedup is very poor, less than 2 in the examples shown. If on the other hand, singleton clusters are rejected (i.e. a cluster must have at least two members) then the work is greatly reduced and super-linear speedup is achieved. The efficacy of this technique depends on the observation that for a typical scene there will be large clusters corresponding to true matches and common geometric alignments (e.g. the common occurrence of right angled planes in our object models),

Table 2. Summary of execution times

Process	Time in seconds 1 processor	Time in seconds 25 processors	Speedup
Canny	23.04	1.12	20.57
Edge Track	15.80	1.01	15.67
HK Calc	581.02	25.35	22.92
HK Segment	45.04	4.53	9.94
Region Grow	673.96	30.51	22.08
Pose (depth)	3.00	0.08	37.5
Pose (intensity)	62.10	0.77	80.65
Total	1403.96	63.37	22.15

but many singleton patterns corresponding to incorrect matches between scene and model features. Provided the cluster size is greater than the number of processors used, it can be shown that thresholding by the worker has a very low probability of rejecting a correct, distributed cluster.

6. Implementation and evaluation of the complete system

Each of the parallel modules which comprise the vision system has been characterised in detail, investigating the effects of variation of parameters and using up to 25 of the total complement of 32 processors. Table 2 summarises the execution times for one and twenty-five processors for the data shown in figure 3. Since the exact execution times are data dependent, these figures can vary for the higher level processes. However, the data given is typical for images of this type and size, 256 by 256 pixels for both depth and intensity data.

The totals given in the final row of table 2 set an upper bound on the possible speedup, and a lower bound on the execution time of the complete parallel process, assuming that 25 processors can be successfully allocated to each task in a sequential manner. This shows a reduction in total processing time from over 23 minutes to just over a minute. The most obvious difference is in the times between the depth and intensity processing routes, of the order of about 12:1 in this case. The HK calculation and region growing processes dominate the work load although the greater simplicity of the 3D-3D matching in the first phase of the clustering process means that pose definition using intensity data takes more time. Since there is a limited number of available processors, this imbalance can make it difficult to balance the two routes to run concurrently. There are also imbalances within the processes of each route; at present each process runs to completion before commencement of the succeeding process. Figure 10 shows the speedup achieved for the segmentation system alone using a quasi-optimal balance between the depth and intensity routes. One intensity processor was used for three to nine depth processors and two intensity processors were used for eleven to twenty-five depth processors. At the top end of this plot, three intensity processors give similar performance to two. Figure 10 includes downloading of the raw images onto the worker processors at the start of computation (but not reading them from files). It also includes several data rearrangements and redistributions throughout the calculation and some post-processing on the resulting data, for instance, calculating

centroids and normals for planes. As a consequence, the performance is much lower than the individual components and is dominated by the major workload modules, particularly the HK calculation routine. Nevertheless, better than an order of magnitude of improvement is achieved using 25 processors on a general purpose parallel MIMD machine, in spite of the imbalance caused by processing the two routes.

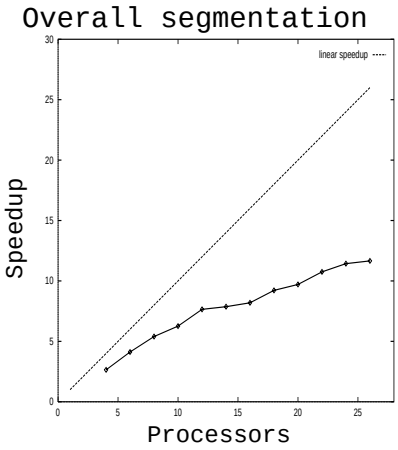


Figure 10. Speedup for overall segmentation system

Given the overall performance of the segmentation system, and of the clustering algorithm presented in Section 5, we were able to carry out a number of tests on image data similar to figure 3. We were restricted to some extent by the field of view of the depth sensor [5] and by the capability of the solid modeller so the real data presented here has only single objects for which registered depth and intensity data could be easily acquired. In addition, we were again restricted to the use of fewer than 32 transputers by the available MIMD architecture. This meant that we could not replicate the overall performance gains shown in table 2 for 25 processors for the individual algorithms. For example, due to the necessity to allocate the majority of the processors to depth segmentation, only three processors were used in the intensity processing route; this meant that the simple geometric decomposition of the Canny algorithm could never allow a speed-up greater than three times. Depth processing was affected greatly by the inclusion of curved (cylindrical) patch processing. Since the depth segmentation is already more complex than the intensity segmentation we ran system tests with and without cylindrical patches.

In addition to the real data, we employed synthetic depth and intensity images generated by a ray tracer, containing up to three objects, e.g. figure 11. This allowed pose estimates to be compared against known ground truth as well as enabling multiple object testing which was difficult to arrange with the real depth sensor. Experiments were also performed which investigated the effects of parameter variation on system performance, but these did not show any systematic trend and are omitted.

Figure 12 shows a timing diagram for system operation which includes processing of cylinders. This was carried out to produce the data of figure 3 using 16 processors for depth

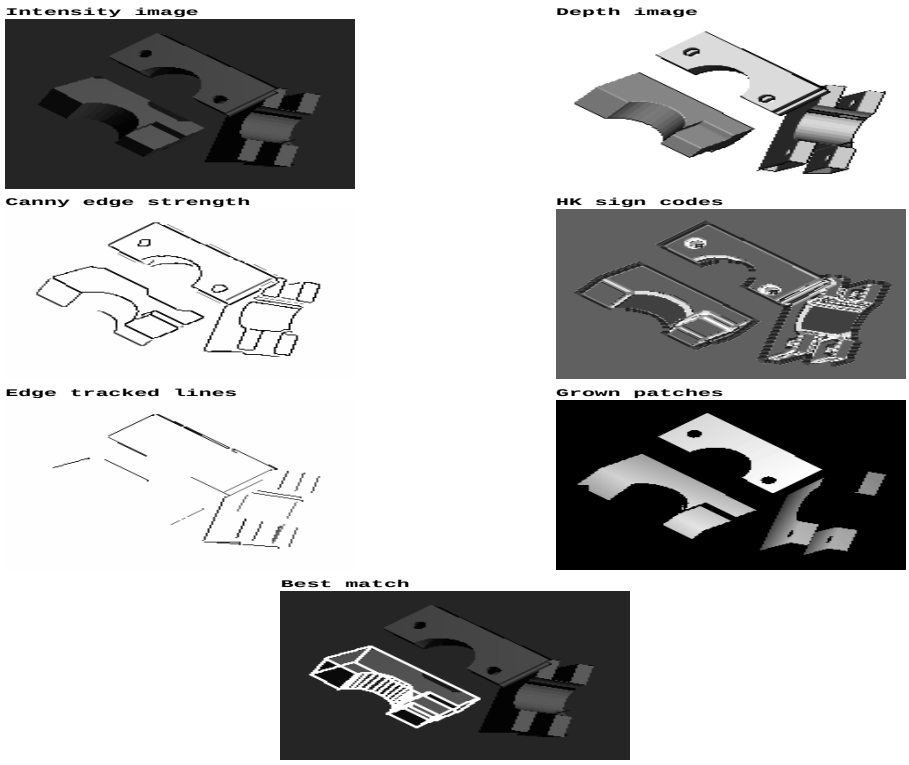


Figure 11. Synthetic image having three objects

segmentation, 3 for intensity segmentation and 4 for pose clustering. Figure 13 shows the timings under the same conditions but excluding cylinder processing. In comparison with table 1, this latter chart shows an increase in processing for the parallel system to just over 80 seconds, an increase in time of approximately 25%. However, time for initialization of data is also included in these charts but not in table 1. The differences between the two figures show how cylinder processing affects the overall performance, since segmentation of the curved surface data requires more computation. Inclusion of cylinders does not significantly affect the pose clustering times in this instance as model cylinders and planes employ the same point-vector representation.

We further investigated the behaviour of the system as the processor-to-process allocation was varied, as shown in Figure 14. The number of intensity processors, `iprocs`, was varied from 1 to 3, the number of depth processors, `dprocs`, from 6 to 18, and for pose clustering either 2 (1x2) or 4 (2x2) were used. In addition, each combination of processor allocations was carried out both including and excluding cylinder processing. For each system run, all the timing and segmentation/clustering data were stored but in Figure 14 the performance is summarised by plotting overall time for segmentation and for pose clustering.

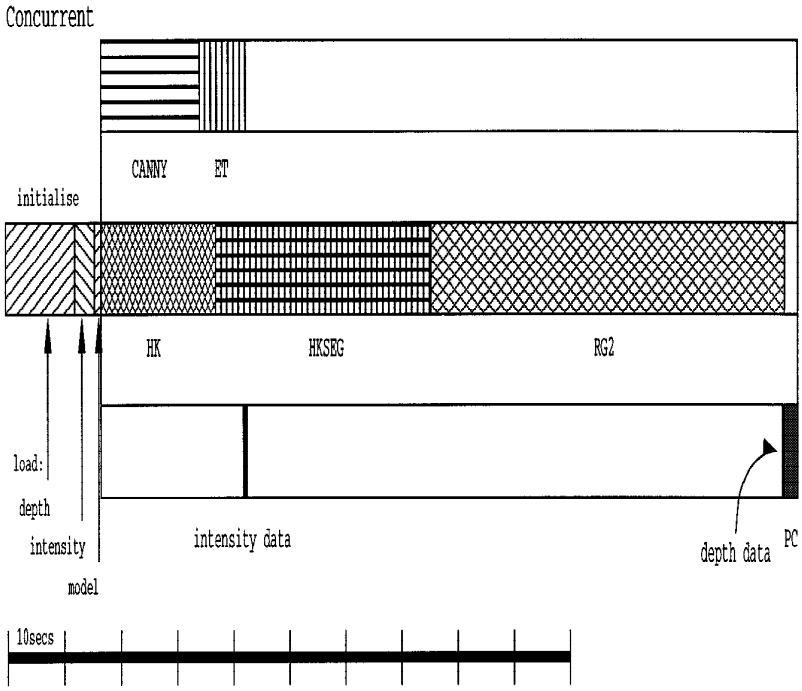


Figure 12. Image 1: Timing diagram for system including cylinder processing

As expected, the configuration of the pose clustering processors (1x2 or 2x2) does not affect the segmentation processing times in the upper part of the diagram. Similarly, the pose clustering times (lower) are not affected by segmentation processor allocation as this is dependent only on the number of segmented features which must be considered. The number of intensity processors affects the overall processing time slightly with about a 20 second reduction from 1 to 2 and about a 10 second reduction from 2 to 3 processors. Use of further intensity processors is counter-productive since this takes processors away from the depth route. Cylindrical processing increases the processing time by about 20 seconds, but in this instance only one cylindrical patch is present. The presence of more cylinders results in a greater increase. In contrast, increasing the depth processors from 6 to 18 reduces the overall processing time by about 70 seconds.

7. Conclusions

We have developed parallel algorithms for low, intermediate and high level processing in a visual system on a general purpose MIMD architecture. The exact speed-up achieved depends on the complexity of the problem. Simple data parallel implementations offer

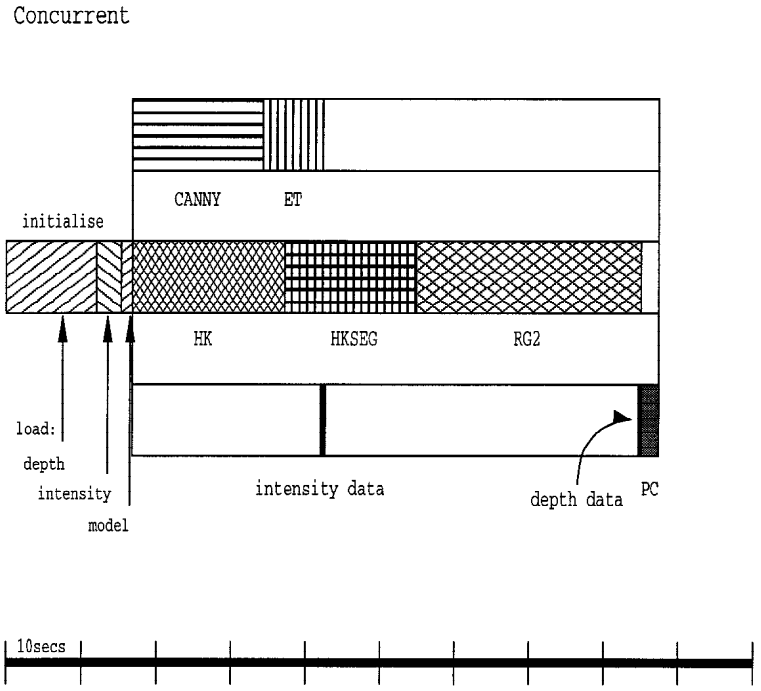


Figure 13. Image 1: Timing diagram for system excluding cylinder processing

near linear speed-up and are comparable to implementations reported elsewhere in the literature. Processes which require inter-worker communication and are not amenable to regular data decomposition, such as edge tracking and region growing, do not provide such easy parallelism but we have devised solutions which still give significant performance gains. Our approach to model-scene matching is based on pose clustering and achieves super-linear speed-up, with the proviso that this depends on a rejection of single data-pattern clusters. This is acceptable provided the number of processors does not exceed the size of a correct cluster substantially, in which case there is a probability that the desired cluster may be lost.

Following our evaluation of the separate algorithms, we then integrated them within a dual-source parallel architecture, so that it was possible to process from input depth and intensity data to identify and locate objects in comparison with stored models. Absolutely, times were slow, relying on processors which have far lower processing speeds than available today. The final system speed-up was between 7:1 and 10:1 in typical cases. This is very strongly dependent on the depth segmentation, since this dominates the processing in figures 12 and 13. Processing single source data can lead to greater speed-up [9], but here we have shown how both routes can be integrated within the same structure. This can increase the

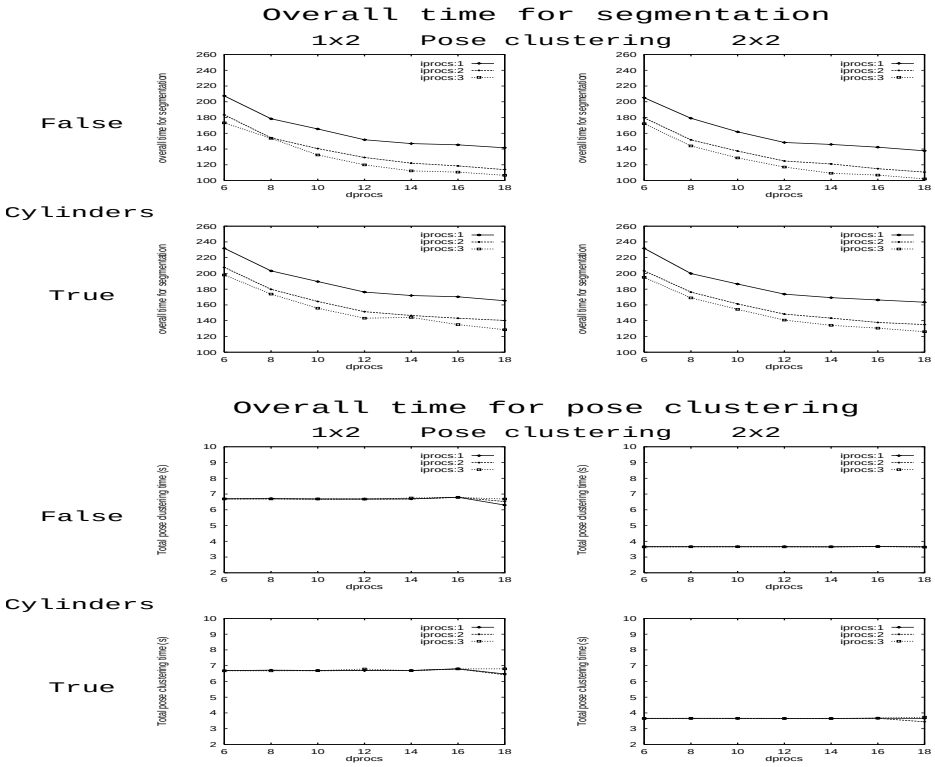


Figure 14. Parallel run time for segmentation and pose clustering

overall robustness of the system since it may not be possible to locate objects on the basis of either single source, but may be possible from both combined.

Results from the integrated system can be compared with the upper bound on system speed-up which was summarised in Table 2 and was of the order of 22:1. The difference arises because of the necessity to split processes amongst available processors according to the architecture illustrated in figure 1 and 2, and because there are initialization and communication costs associated with control of the system as a whole which are not present when considering single algorithms in isolation. In absolute terms, the single processor times were not dissimilar from comparable implementations in C on Sun4 workstations, and the principal issue is the speed-up achieved and the generality of the approach.

Therefore, the principal contribution of this work has been to show the parallel implementation of a complex visual system on a general purpose MIMD architecture. Separate algorithms have exhibited scalable performance within the limits of the available processors, and we anticipate that this would scale much further. System evaluation has shown the difficulties imposed by the limitations of the machine and language, yet significant speed-up has been achieved. Given the performance of the separate algorithms and experience gained from the whole system, we contend that it is possible to build fast, parallel vision

systems on such an architecture, giving concern to the necessity for sufficient memory to allow dynamic rather than static allocation of processes to processors, and the provision of dynamic data structures to further optimize memory usage. With recent increases in processor speed of almost two orders of magnitude when compared to the T800 units, the prospect of *real time*, as opposed to *reasonable* performance becomes apparent.

Acknowledgments

This work was supported by ESPRC grant number GR/J07884.

References

1. W.J. Austin and A.M. Wallace. Recognition and location by parallel pose clustering. In *Proc. of 6th British Machine Vision Conference*, pages 661–670, 1995.
2. P.J. Besl and R.C. Jain. Invariant surface characteristics for 3d object recognition in range images. *Computer Vision, Graphics and Image Processing*, 33:33–80, 1986.
3. J. F. Canny. A computational approach to edge detection. *IEEE Trans. PAMI*, 8(6):679–698, 1987.
4. V Chaudhary and J.K. Aggarwal. Parallelism in computer vision: A review. In V. P. Kumar, P. S. Gopalakrishnan, and L. N. Kanal, editors, *Parallel Algorithms for Machine Intelligence and Vision*. Springer-Verlag, 1990.
5. J Clark, G Zhang, and A.M. Wallace. Image acquisition using fixed and variable triangulation. In *Proc. of IEE Int. Conf on Image Processing*, pages 539–543, 1995.
6. M. I. Cole. *Algorithmic skeletons: structured management of parallel computation*. Pitman, 1989.
7. M.A. Fischler and R.C. Bolles. The random sample consensus set: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
8. V. Leavers. Which Hough transform. *CVGIP: Image Understanding*, 58(2):250–264, 1993.
9. G.J. Michaelson and N. Scaife. Prototyping a parallel vision system in Standard ML. *Journal of Functional Programming*, 5(3):345–382, 1995.
10. M. Rygol, S.B. Pollard, and C.R. Brown. MARVIN and TINA: A multiprocessor 3d vision system. *Concurrency: Practice and Experience*, 34(4):333–356, 1991.
11. M. Rygol, S.B. Pollard, and C.R. Brown. Multiprocessor 3d vision system for pick and place. *Image and Vision Computing*, 9(1):33–38, 1991.
12. G. Stockmann. Object recognition and localization via pose clustering. *Computer Vision, Graphics and Image Processing*, 40:361–387, 1987.
13. E. Thirion and A.M. Wallace. Compilation and sufficient representation of object models for visual recognition. *Pattern Recognition Letters*, 13:797–803, 1992.
14. R. Y. Tsai. A versatile camera calibration technique for high accuracy 3d machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Trans. on Robotics and Automation*, 3(4):323–344, 1987.
15. A.M. Wallace, G.J. Michaelson, P. McAndrew, K. Waugh, and W. Austin. Dynamic control and prototyping of parallel algorithms for intermediate and high level vision. *IEEE Computer*, 25(2):43–53, February, 1992.
16. C-L. Wang, P.B. Bhat, and V.K. Prasanna. High performance computing for vision. *Proceedings of the IEEE*, 84(7):931–945, 1996.
17. H. Wang and M. Brady. PARADOX — a heterogeneous machine for the 3d vision algorithm. *Concurrency: Practice and Experience*, 7(4):257–272, 1995.
18. C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, D.B. Shu, and G. Nash. The image understanding architecture. *International Journal of Computer Vision*, 2:251–282, 1989.
19. S. Yalamanchili and J.K. Aggarwal. Parallel processing methodologies for image processing and computer vision. *Advances in Electronics and Electron Physics*, 87:259–299, 1994.