②

RADC-TR-90-404, Vol XIII (of 18)
Final Technical Report
December 1990

IMAGE UNDERSTANDING AND INTELLIGENT PARALLEL SYSTEMS

Northeast Artificial Intelligence Consortium (NAIC)

Christopher M. Brown

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded partially by the Laboratory Director's fund.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-404, Volume XIII (of 18) has been reviewed and is approved for publication.

APPROVED: Lee A. Uvanni

LEE A. UVANNI
Project Engineer

APPROVED: Garry W. Barringer

GARRY W. BARRINGER
Director of Special Programs
Directorate of Intelligence & Reconnaissance

FOR THE COMMANDER: Igor G. Plonisch

IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRRE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 1990	3. REPORT TYPE AND DATES COVERED Final Sep 84 - Dec 89

4. TITLE AND SUBTITLE
IMAGE UNDERSTANDING AND INTELLIGENT PARALLEL SYSTEMS

5. FUNDING NUMBERS
C - F30602-85-C-0008
PE - 62702F
PR - 5581
TA - 27
WU - 13
(See reverse)

6. AUTHOR(S)
Christopher M. Brown

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Northeast Artificial Intelligence Consortium (NAIC)
Science & Technology Center, Rm 2-296
111 College Place, Syracuse University
Syracuse NY 13244-4100

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Center (COES)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-404, Vol XIII
(of 18)

11. SUPPLEMENTARY NOTES
(See reverse)
RADC Project Engineer: Lee A. Uvanni/IRRE/(315) 330-4863

This effort was funded partially by the Laboratory Director's fund.

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

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose was to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress during the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photointerpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system.

The specific topics for this volume are various aspects of parallel, structural and optimal techniques in computer vision.

14. SUBJECT TERMS
Artificial Intelligence, Parallel Computation, Edge Detection, Computer Vision, Multiprocessing

15. NUMBER OF PAGES
80

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	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Block 5 (Cont'd)   Funding Numbers

PE - 62702F	PE - 61102F	PE - 61102F	PE - 33126F	PE - 61101F
PR - 5581	PR - 2304	PR - 2304	PR - 2155	PR - LDFP
TA - 27	TA - J5	TA - J5	TA - 02	TA - 27
WU - 23	WU - 01	WU - 15	WU - 10	WU - 01

Block 11 (Cont'd)

This effort was performed as a subcontract by the University of Rochester to Syracuse University, Office of Sponsored Programs.

VOLUME 13
Final Technical Report
Image Understanding and Intelligent Parallel Systems
Contract F30602-85-C-0008

Christopher M. Brown, P.I.
Computer Science Department
University of Rochester
Rochester, NY 14627

August 7, 1990

Contents

1 Overview

Under this RADC contract the University of Rochester developed and disseminated papers, ideas, algorithms, analysis, software, applications, and implementations for parallel vision applications and programming environments for parallel computer vision. The work has been widely reported and highly influential. The investigators have been awarded several honors Faculty members involved have received several prestigious honors, including an IBM Faculty Development Award for Michael Scott and an ONR Young Investigator Award for Tom LeBlanc. We were awarded a DARPA Parallel Systems postgraduate fellowship. We have won several Best Paper awards. From 1984 to 1989 the department produced approximately 400 papers, more than half of which are in refereed conferences and journals. There have been 14 completed Ph.D. theses directly related to parallel vision and the related programming environment, and approximately ten more such theses are in progress.

As a part of the RADC contract, we developed a heterogeneous parallel architecture involving pipelined and MIMD parallelism, and integrated it with a high performance 9 degree of freedom robot head. The hardware of the laboratory is described in the next section. The most significant environment development work centered on the Butterfly Parallel Processor and the MaxVideo pipelined parallel image processor. For the Butterfly, the Psyche multi-model operating system was developed (as well as two other experimental operating systems), and the Lynx language compiler ported. Much basic and influential performance monitoring and debugging work was completed, resulting in working systems and novel algorithms. There was also significant research in systems and applications using the other parallel architecture in the laboratory, the MaxVideo parallel pipelined image processor.

Early in the contract period, Rochester demonstrated SIMD-like programs on the BBN Butterfly Parallel Processor that show linear parallel speedup. Many applications for the image processing pipeline (including tracking, color histogramming, feature detection, frame-rate depth maps, frame-rate time-to-collision maps, large-scale correlations, segmentation using motion blur, and others) have been written. The efficacy of intimate cooperation between vision computations and controlled motion has been demonstrated. This work has attracted national attention and won international prizes. The Zebra object-oriented system for Datacube programming was developed, and the Zed menu editor built on top of Zebra. These programming environments are useful for any register-level devices, and are a considerable improvement on previous Datacube environments. They are being made available to all by anonymous ftp.

Programming MIMD applications is difficult, and Rochester is a leader in developing operating systems (PSYCHE), performance monitoring (PPUTTS) and debugging (INSTANT REPLAY) tools to make the job easier. The PLATINUM system solves automatically many of the problems (code and data replication and cacheing) in getting SIMD-like programs to run efficiently on Non Uniform Memory Access architectures (such as hypercubes, Butterfly, Encore, etc.). The MIMD program development tools (PPUTTS, Instant Replay, and Moviola) provide several graphical views and a LISP interface to a multi-process, multi-processor application. The system provides repeatable single-stepping, statistics, symbolic debugging, and other "traditional" debugging techniques that have not previously been available to parallel programmers. This work has produced many influential papers, several prizes, and the operational systems.

At the end of the contract period the PSYCHE operating system was operational, and is currently supporting multi-agent applications, and multi-model (e.g. both threads and heavyweight processes) programming environments. PSYCHE has been used to support five independent processes controlling the bouncing of a tethered balloon with a paddle – this hybrid system uses pipelined parallelism from the MaxVideo system for low level visual input. As a result of the RADC contract, we are now developing plans (the ARMTRAK system) for integrating pipelined parallelism, MIMD parallelism with multiple computational models and sequential planning paradigms to manage a dynamic model railroad system.

Rochester has implemented object recognition algorithms in neural nets, and developed hardware realizations for the resulting constraint-propagation networks. The domain includes large sets of objects, and uses Bayesian techniques to handle partial and incomplete information. The Rochester Connectionist Simulator and the Zebra/Zed systems are available by anonymous ftp. Together they have been distributed to several hundred sites worldwide.

This final report starts with a quick guide to key papers that have been produced over the years, and then in turn briefly outlines the Laboratory, parallel computer vision applications, integration of a cognitive layer into the system, support work in operating systems, languages, utilities, performance monitoring, pipelined parallelism, and technology transfer issues. A list of theses produced under the contract is included. More detail is available from the papers in the literature, and extensive references are provided.

2 Key Reports by Topic

This section briefly points out key reports. More detail on these projects appears in later sections of this final report.

2.1 Laboratory for Parallel Vision Research

During the contract period, Rochester developed and commissioned a binocular robot head, acquired and commissioned a multiple degree-of-freedom platform for the 3-dof robot head, and acquired a real-time, pipelined parallel image processing engine.

The laboratory allows us to test our systems concepts in a complex, visuo-motor real-time environment. Software integration is important as well: PSYCHE's first application will be to manage the higher-level data structures (e.g. the world model) in an integrated parallel vision system that also uses the pipelined parallelism of the frame-rate MaxVideo image processing system. The key reports are [Brown et al. 1988 (Rochester Robot); Ballard 1990 (Animate Vision); Ballard et al. 1987 (Eye Movements); Brown and Rimey 1988 (Coordinate systems, kinematics...); Brown 1988 (Parallel Vision with the Butterfly); Brown 1989a (Gaze Control)].

2.2 Vision Applications

Vision applications are an important part of our work, but are only indirectly supported by the contract, which views applications as potential users of the parallel systems we are developing. For example, Paul Chou's work used the Markov Random Field formulation for intermediate-level vision and produced results that have been quantified and are better than any other known techniques. We have ported his evidence-combination to the Butterfly, where it runs as a set of three cooperating agents under Tom LeBlanc's SMP system. As another example, the work of Cooper and Swain is being ported to the Connection Machine at the University of Syracuse's Parallel Computing Facility, NPAC. Object recognition, inference, quantification of performance in biologically oriented neural net computational techniques, and hardware for relaxation computations have all been under active study.

Several parallel vision applications were pursued, including Butterfly programming, Markov Random Field and connectionist research, and work aimed at integrating the real-time laboratory and using it for complex planning tasks that include sensing and acting. Key papers are [Feldman et al. 1988a,b; Feldman 1987 (Basic connectionism); Simard et al. 1988 (Recurrent backpropagation); Porat and Feldman 1988 (Learning theory); Olson et al. 1987 (Vision on butterfly); Ballard and Ozcandarli 1988 (Kinetic depth calculations); Brown et al. 1989a (decentralized Kalman filters); Aloimonos and Brown 1988 (Robust computation of intrinsic images); Chou and Brown 1988 (Sensor fusion, reconstruction and labeling); Wixson and Ballard 1990 (Color histograms); Rimey and Brown 1990 (Hidden Markov models); Yamauchi 1989 (Juggler); Nelson 1990 (Flow fields); Cooper 1988 (Structure recognition); Sher 1987a,b,c (Probabilistic low-level vision.); Swain 1988 (Object recognition from large database); Swain and Cooper 1988 (Parallel hardware for recognition); Martin, Brown, and Allen 1990 (ARMTRAK project); Allen and Hayes 1985 (Theory of time), Allen 1989 (Representing time)].

2.3 Parallel Hardware and Programming Languages

Throughout the contract period Rochester has kept pace with the technical developments of the Butterfly product line of BBN-ACI. We have owned three generations of Butterfly computers, including one of the largest ever sold. Much of our research transcends any particular piece of hardware, though its implementation of course requires intimate familiarity with particular hardware.

Languages for MIMD parallel computers have been developed and ported under the contract, and quantitative comparisons made between programming models. A library for programming the MaxVideo pipeline parallel image analysis hardware has also been developed. The key reports are [LeBlanc et al. 1988 (Large-scale parallel programming); Scott et al. 1990 (Multi-model parallel programming); Crowl 1989 (A uniform object model); Tilley 1989 (Zebra for MaxVideo)].

2.4 Parallel Programming Environment – Operating Systems

Three operating systems (Elmwood, Platinum, Psyche) have been developed for the Butterfly. The most ambitious project is Psyche, though Platinum solves automatically a number of problems that users face when using Uniform System-style programming on a MIMD computer (Automatic cacheing and data migration, for instance). The key papers are [Scott et al. 1989b,c (Psyche description); LeBlanc et al. 1989b (Elmwood description); Cox and Fowler 1989 (Platinum description)].

2.5 Parallel Programming Environment – Utilities and Libraries

Along with languages and operating systems, Rochester produced systems utilities for communication, file systems, and compilers. They span a broad range from parallel file systems through new languages for expressing parallel computation. Applications packages such as the current version of the neural net simulator and the image-processing utilities allow speedups of up to a factor of 100 over single-workstation implementations. User interfaces to large multiprocessor computers are a difficult issue addressed by Yap's work, and many of the packages extend the range of computational models available to a user. For instance, the Ant Farm project provides capability we noticed we needed after the first DARPA Parallel Architectures Benchmark and Workshop, namely the ability to support many lightweight processes. The key papers are [Scott and Jones 1988 (Ant Farm); Dibble and Scott 1989a,b (Bridge file system); Bolosky et al 1989 (memory management techniques); Goddard et al. 1989 (Connectionist simulator); LeBlanc and Jain 1987 (Crowd control); Yap and Scott 1990 (PenGuin)].

2.6 Parallel Programming Environment – Performance Monitoring

Debugging and performance monitoring in an MIMD environment are significantly more difficult than on a uniprocessor. Rochester contributed many results over the course of the contract. The instant replay system allows normal cyclic debugging in a nondeterministic parallel environment by keeping a log of interactions between processes. Moviola is a suite of interactive performance monitoring tools. The key papers are [LeBlanc and Mellor-Crummey 1987 (Instant Replay); Fowler et al. 1988, LeBlanc et al. 1990 (Moviola)].

3 The Laboratory

The Rochester Robotics Laboratory has developed, during the years of the RADC contract, to the configuration described in this section. It currently consists of four key components (Fig. 1): a "head" containing cameras for visual input, a robot arm that supports and moves the head, a special-purpose parallel processor for high-bandwidth, low-level vision processing, and a general-purpose parallel processor for high-level vision and planning. This unique design allows for visuo-motor exploration over an 800 cubic foot workspace, while also providing huge computing and power resources. Thus, we do not suffer the communication and power limitations of most mobile platforms.

The robot head (shown in Fig. 2) built as a joint project with the University's Mechanical Engineering Department, has three motors and two CCD high-resolution television cameras providing input to a MaxVideo digitizer and pipelined image-processing system. One motor controls pitch or altitude of the two-eye platform, and separate motors control each camera's yaw or azimuth, providing independent "vergence" control. The motors have a resolution of 2,500 positions per revolution and a maximum speed of 400 degrees/second. The controllers allow sophisticated velocity and position commands and data read-back.

The robot body is a PUMA761 six degree-of-freedom arm with a two meter radius workspace and a top speed of about one meter/second. It is controlled by a dedicated LSI-11 computer implementing the proprietary VAL execution monitor and programming interface.

The MaxVideo system consists of several independent boards that can be cabled together to achieve many frame-rate image analysis capabilities: digitizing, storage, and transmission of images and sub-images, 8x8 or larger convolution, pixel-wise image processing, cross-bar image pipeline switching for dynamic reconfiguration of the image pipeline, look-up tables, histogramming and feature location. A digital signal processing computer on one board can perform arbitrary computations, and also has a high speed image bus interface and a VME bus master interface so it can

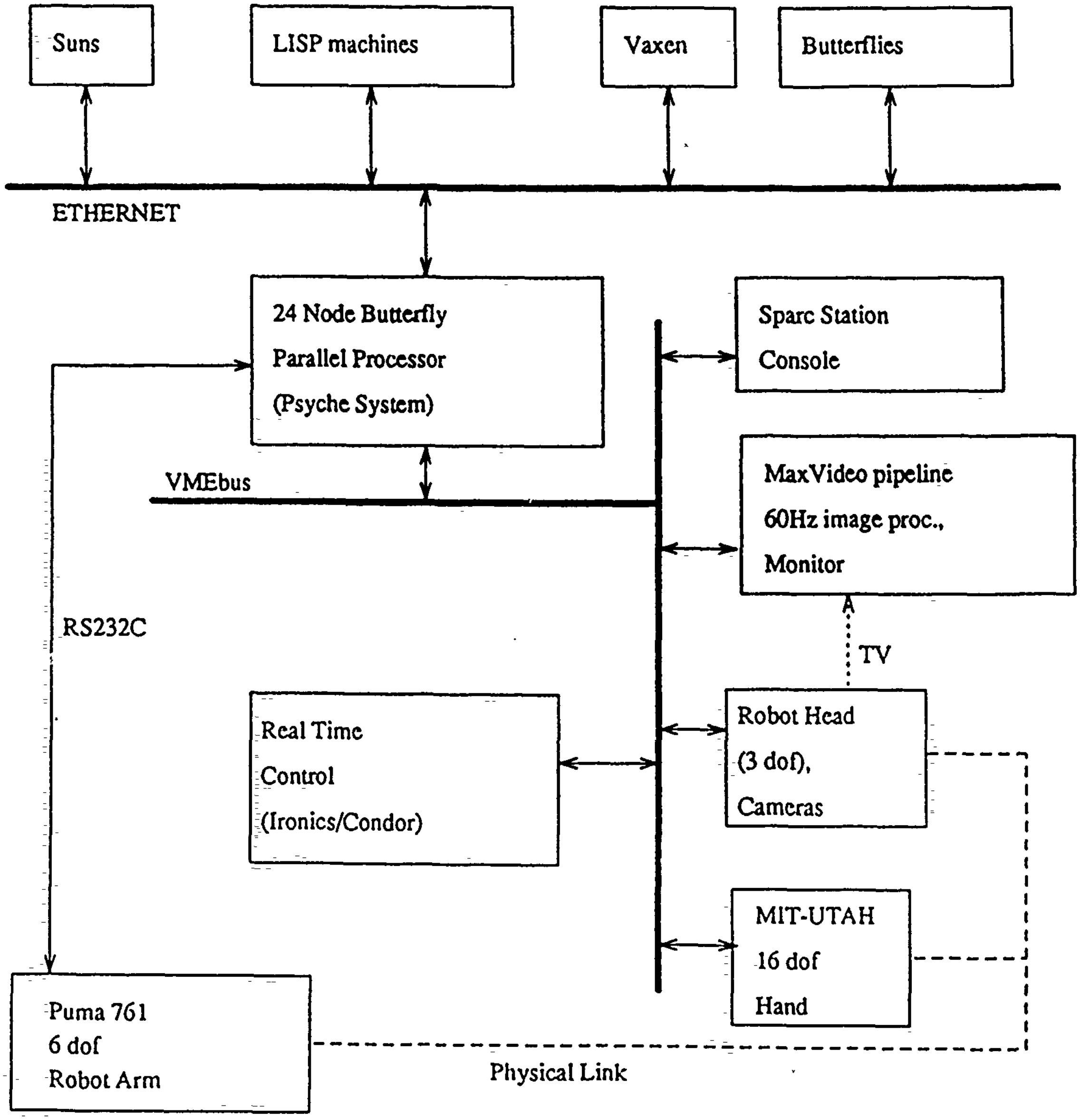


Figure 1: Robotics Laboratory Hardware

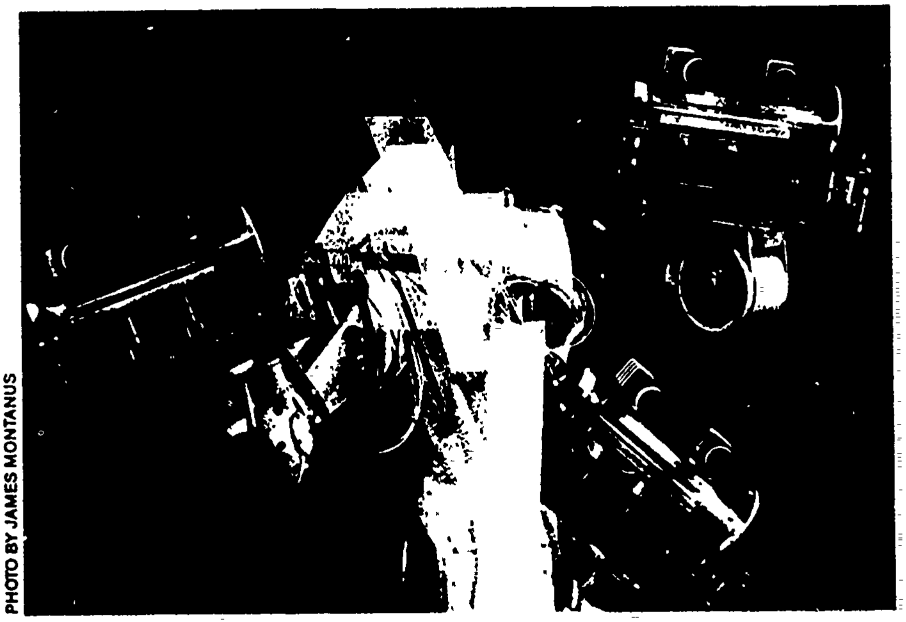


Figure 2: The Rochester Robot. A multi-exposure photograph of the "Rochester Robot" in action. The arm is the largest industrial arm on the market, while the unique head was designed by Professor Dana Ballard.

program the other boards in the same manner as the host. The MaxVideo boards are all register programmable and are controlled by the Butterfly or Sun via VME bus.

A unique feature of our laboratory, one crucial for our future research, is the capability to use a multiprocessor as the central computing resource and host. Our Butterfly Plus Parallel Processor contains 28 nodes, each consisting of an MC68020 processor, MC68851 MMU, MC68881 FPU, and 4 MBytes of memory. The Butterfly is a shared-memory multiprocessor with non-uniform memory access times; each processor may directly access any memory in the system, but with approximately 15 times greater latency. The Butterfly has a VME bus connection that mounts in the same card cage as the MaxVideo and motor controller boards. Currently, a SUN workstation acts as a host system for the lab. As software develops on the Butterfly, we plan to migrate functionality from the workstation host to the Butterfly.

The RADC contract supported the development of parallel applications algorithms and the development of software for the two parallel computing engines in this laboratory, the Butterfly and the MaxVideo.

4 Parallel Vision Applications

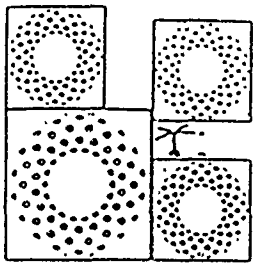
Although the focus of the contract was on developing a programming environment, Rochester also did parallel vision applications as a test and a driving force for the systems development. This section briefly outlines some of the more influential of the projects: more details are available in the literature [e.g. Brown et al. 1985; 1988].

4.1 SIMD-style Low-level Vision on the Butterfly

Rochester participated in the first DARPA benchmark study. One aspect of that work motivated much of our current research in multi-model parallel programming environments and performance modeling tools. The other aspect was a successful demonstration that SIMD-style (data-parallel) low-level vision applications could be performed on an MIMD computer. Fig. 3 shows some results for border-following. Extensive analysis and demonstration programs for multi-resolution image pyramid generation, line finding, connected component analysis, and the Hough transform were also developed [Brown 1986; Olson 1986b,c; Olson et al 1987].

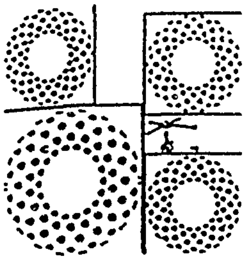
4.2 Parallel Object Recognition

Paul Cooper and Michael Swain cooperated to investigate object recognition, based on object relational structure and some geometry, from a large database. This work was based in connectionist, massively parallel framework, and led to hardware (VLSI circuit) designs and implementations on the connection machine at NPAC in Syracuse,

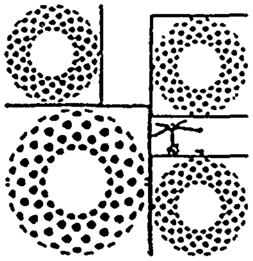


input image

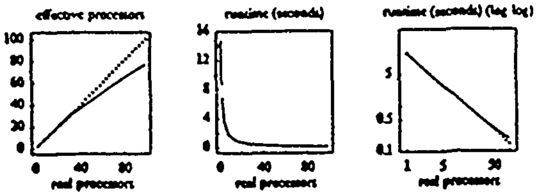
border following runtime statistics			
effective processors	processor efficiency	runtime (seconds)	
*	*	5.228	initialization for benchmark (100 processors)
2.49	2.4%	.653	* Bgetimage & share (512 x 512)
.72	.7%	.680	* allocsm & share
84.28	84.2%	6.950	convolve
72.10	72.1%	.100	detect zero crossings
.61	.6%	.181	* uf_init
64.83	64.8%	.299	label
77.06	77.0%	.193	neighbor
58.05	58.0%	.557	merge
25.63	25.6%	.036	tally (lines found: 1182 too short, 1564 long enough (2 pixels))
58.34	58.3%	.311	relabel & SM_write
.21	.2%	.036	* FreeAll
* (serialized by Uniform System memory allocator)			



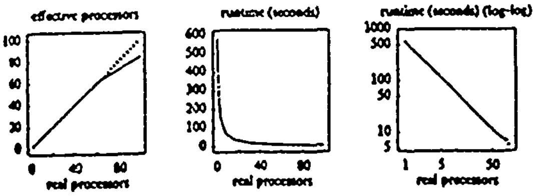
after Laplacian and zero-crossing detection: selected edges



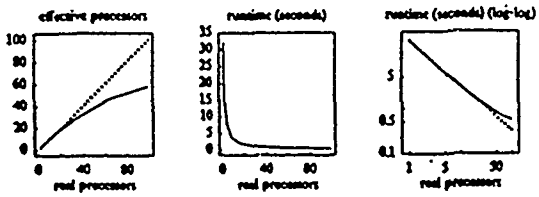
after border following: omit edges on borders shorter than threshold length



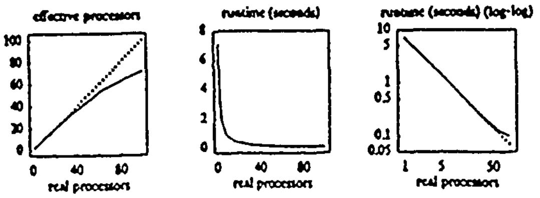
border following benchmark: neighbor



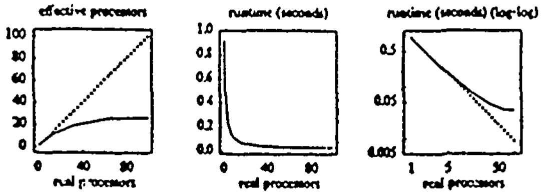
border following benchmark: 11 x 11 Laplacian



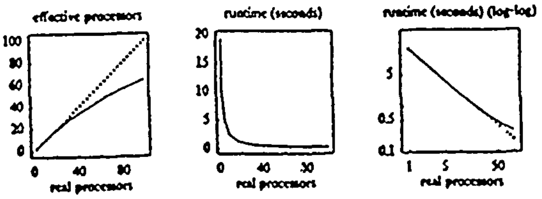
border following benchmark: merge



border following benchmark: zero crossings



border following benchmark: tally



border following benchmark: label

Figure 3: Speedup results of boundary following algorithms on the Butterfly.

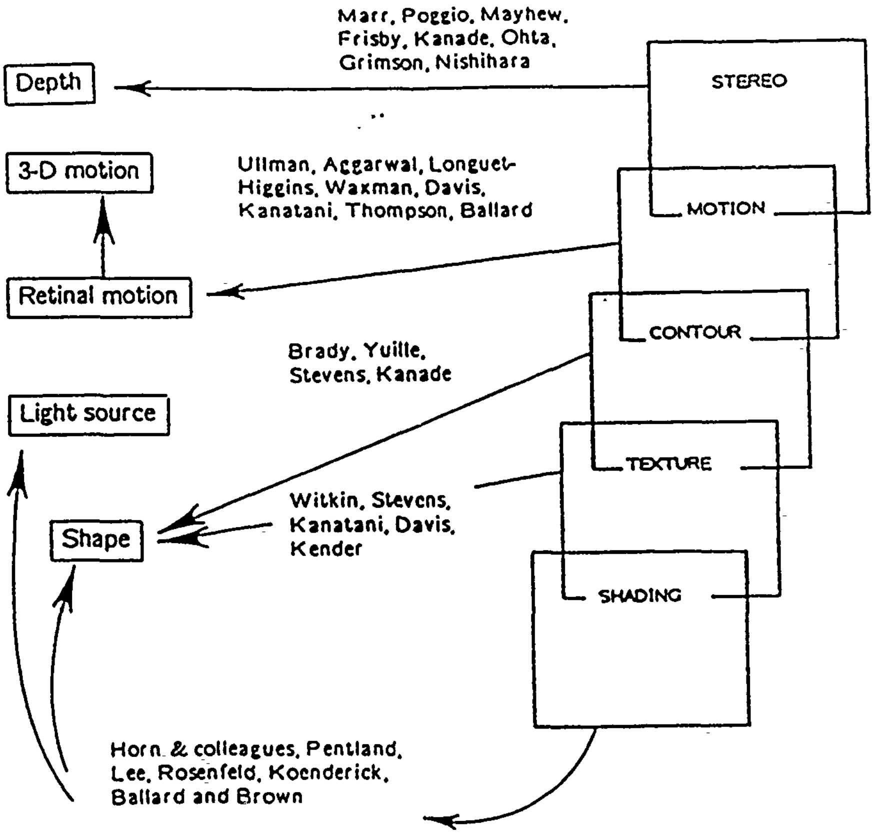


Figure 4: Previous work on intrinsic image calculation.

NY [Cooper 1988, 1989; Cooper and Swain 1988, 1989; Swain and Cooper 1988; Swain 1988].

4.3 Cooperating Intrinsic Image Calculations

John Aloimonos took a mathematical approach in his thesis to unifying several disparate results on extracting physical attributes from images [Aloimonos et al. 1985, Aloimonos 1986; Aloimonos and Brown 19841,b, 1988, 1989; Aloimonos and Swain 1985; Brown et al. 1987, etc.]. The state of knowledge when he started is shown in Fig. 4.

As a result of his work, mathematical constraints were developed to allow these calculations to be combined to produce more robust results with less restrictive assumptions. This work is reported in his recent book *Integration of Visual Modules*, written with Dave Schulman, and summarized in Fig. 5.

The characteristics of well-known visual problems are radically changed by this approach (Fig. 6), which yields robust, linear solutions with fewer assumptions.

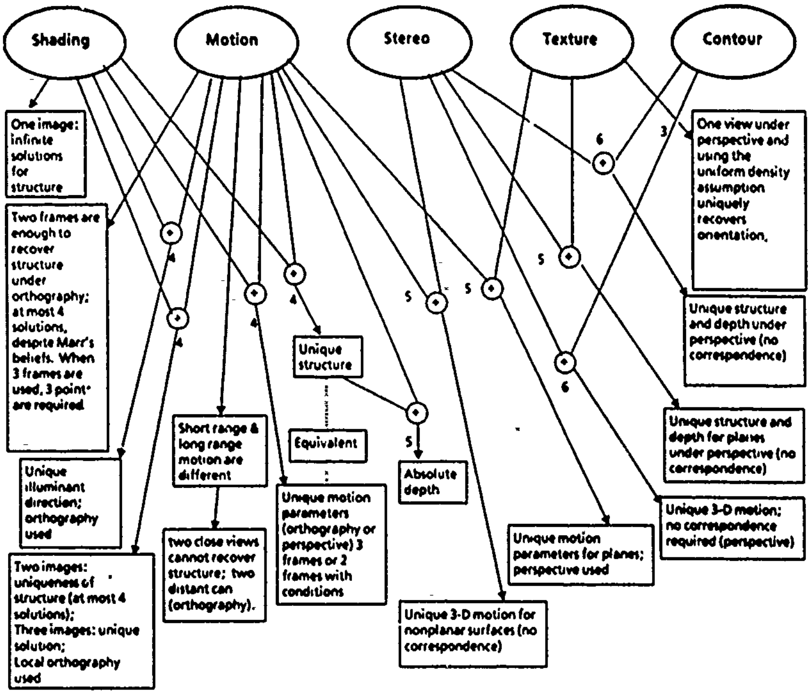


FIGURE 2.4.

Figure 5: Some of Aloimonos' contributions in cooperating intrinsic image calculation.

Problem	Passive Observer	Active Observer
Shape from shading	**Ill-posed problem. Needs to be regularized. Even then, unique solution is not guaranteed because of nonlinearity.**	**Well-posed problem. Unique solution. Linear equation used. Stability.**
Shape from contour	Ill-posed problem. Has not been regularized up to now in the Tichonov sense. Solvable under restrictive assumptions.	Well-posed problem. Unique solution for both monocular or binocular observer.
Shape from texture	Ill-posed problem. Needs some assumption about the texture.	Well posed problem. No assumption required.
Structure from motion	Well posed but unstable. Nonlinear constraints.	Well posed and stable. Quadratic constraints, simple solution methods, stability.
Optic flow (area based)	Ill-posed. Needs to be regularized. The introduced smoothness might produce erroneous results.	Well posed problem. Unique solution. Might be unstable.

Figure 6: Combining constraints gives better solutions for vision problems.

4.4 Markov Random Fields and Massively Parallel IU

In their thesis work, Dave Sher and Paul Chou pursued a probabilistic approach to image understanding, which could be implemented as a Markov Random Field [Sher 1987a,b,c; Chou 1988, Chou and Brown 1987a,b, Chou et al. 1987; etc.]. Image understanding then takes the form of labelling individual pixels or features in the image with properties such as "boundary", "no boundary", or a depth value. This approach allows for a uniform and real-time evidence combination algorithm for multi-sensor fusion, and a parallelizable algorithm for the labelling. Using this approach, the reconstructionist visual approach that tries to create depth maps from images is integrated with the solution to the segmentation problem, which identifies boundaries and objects within the scene. Chou developed the Highest Confidence First algorithm for labelling. Chou made quantitative comparison between several known Markov Random Field algorithms, and HCF was shown to be a superior method to all those known at the time. HCF is inherently sequential. Later work at Rochester by Swain and Wixson parallelized the algorithm for the Butterfly, with improved qualitative, quantitative, (and of course timing) results [Swain and Wixson 1989, Swain et al. 1989].

Fig. 7 shows the performance of HCF on a boundary-detection task.

Fig. 8 shows the results of combining sparse depth measurements with intensity data to produce a depth map of the scene and a boundary map simultaneously.

4.5 Pipelined Parallelism and Real-time Object Search

A good example of the cooperation of real-time vision processing and a mobile observer is provided by Rochester's program of work on fast object detection, which uses relational modeling, and reasoning about occlusion.

The ability to find a certain object in an unknown environment is a component of many real-world problems that a general-purpose robot might face. Lambert Wixson studied this visual task, *object search.* His research is divided into three areas, all of which attack the key problem of robustly finding the object in the smallest possible time.

The first is the problem of object recognition. Most research on model-based object recognition from a single camera has concentrated on robustness. While this is obviously an important first step, the object search task brutally illustrates that speed is just as important. Almost all current object recognition schemes require that image features be matched to model features, requiring a time polynomial in the number of features to perform the matching. This polynomial time is a result of having to match the image features to the model features in order to calculate and refine the pose estimate of the object in the scene. By adding an initial stage that does not perform pose calculation but rather simply detects the likely presence of

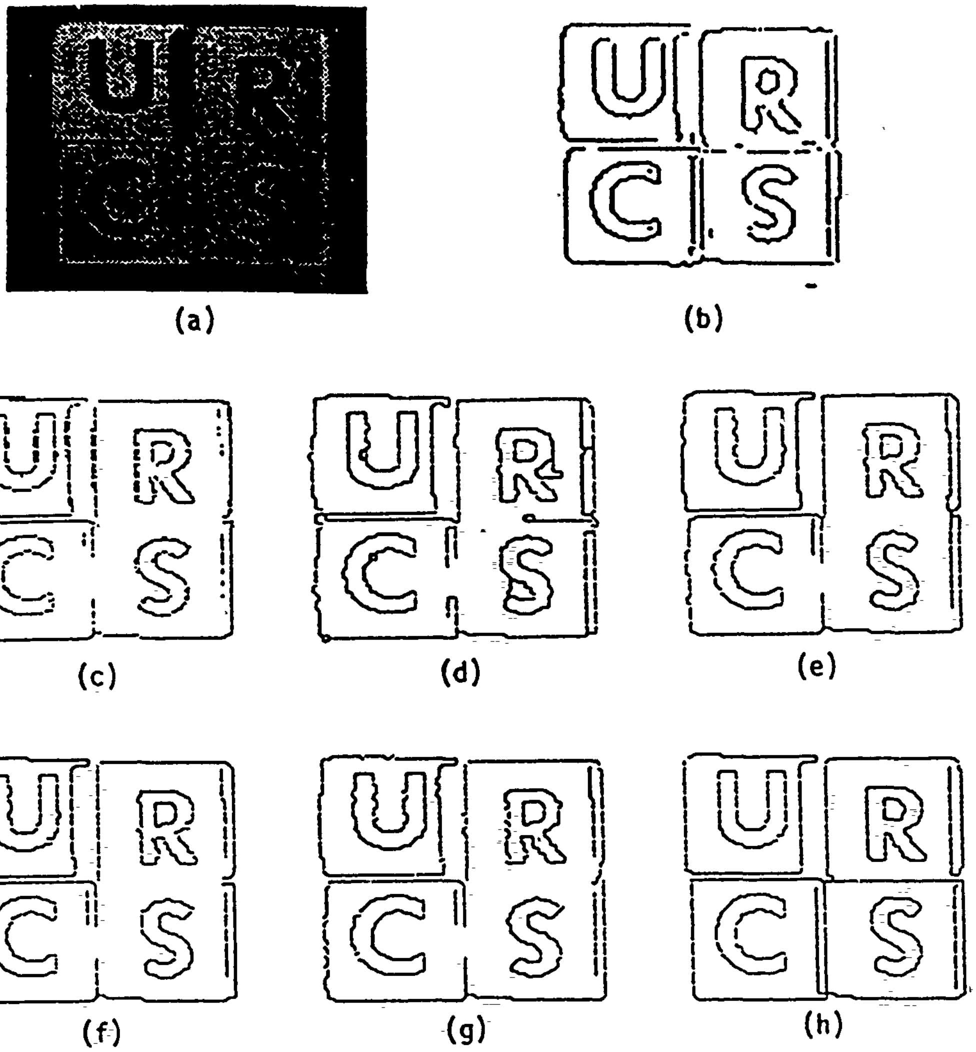


Figure 7: Highest Confidence First algorithm and edge-finding. Boundary detection experiment set (III).

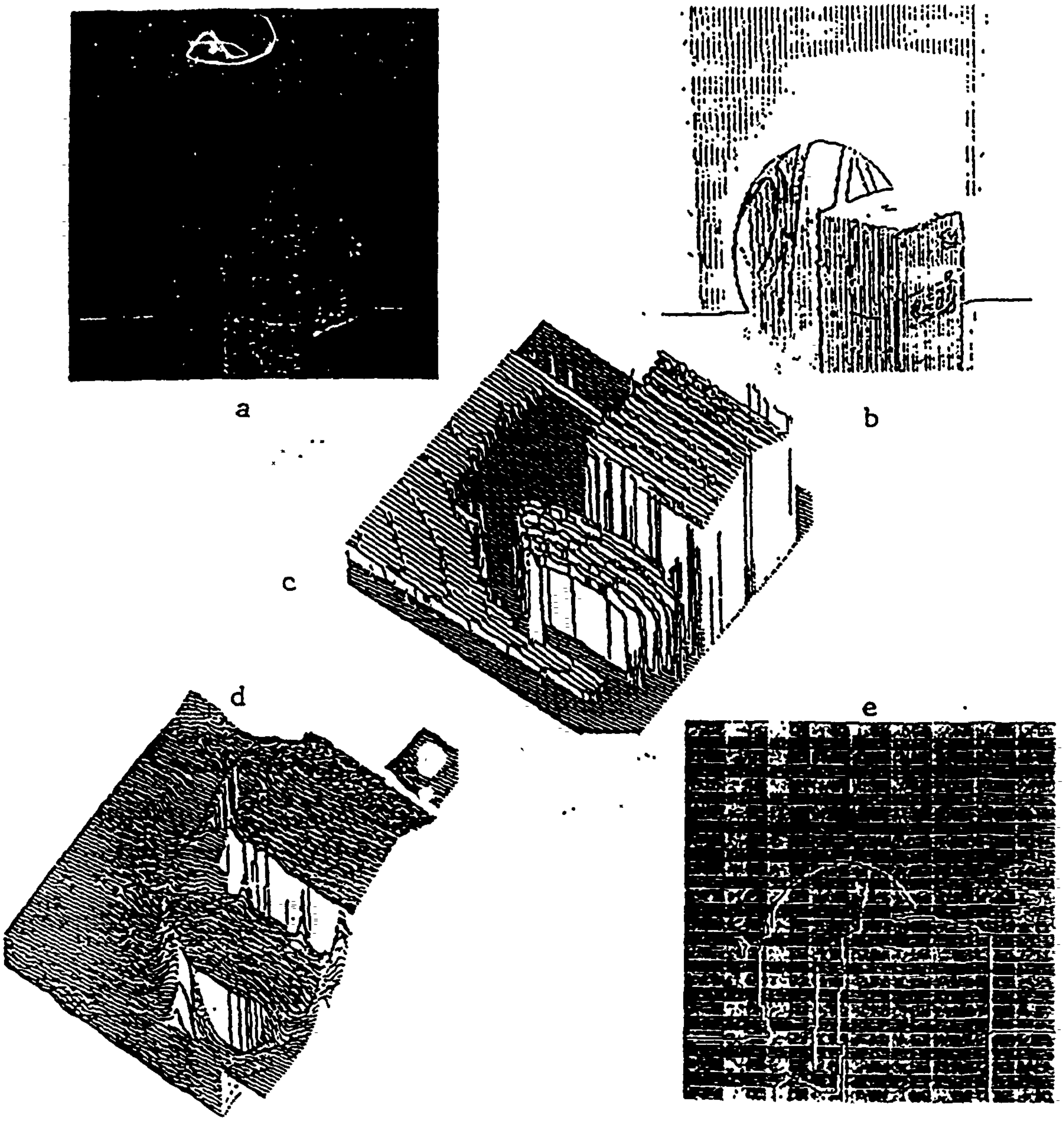


Figure 8: Highest Confidence First algorithm, segmentation, and depth boundary calculation. Experiments with stereo disparity data (II).

the object in the image, considerable efficiency can be gained. The idea is that this initial stage would be used to rank each gaze in a set of candidate gazes according to the likelihood that the image produced by the gaze contains the desired object. This ranking can then be used to choose the order in which a more sophisticated object recognition program (which would calculate pose) should be applied to the candidate images.

Wixson [Wixson and Ballard 1989] constructed an *object detection* scheme that relies on the assumption that the color histogram of an object can be used as an object "signature" which is invariant over a wide range of scenes and object poses. The color histogram is computed at 3Hz by the Datacube hardware, and the matching compares 18 database items to a histogram in one second. Counting time to move the robot to a new gaze position (one and one-half seconds per move), each gaze can be evaluated for its object content in just under 3 seconds. Fig. 9 shows some sample results.

The second area of object search is the use of high-level knowledge of common relationships and interactions between objects (*i.e.*the contexts in which certain objects typically appear) to direct the search process [Wixson to appear]. For example, if the robot is looking for a pen, it might be wise to search for a desk first. referred to this use of high-level knowledge as *indirect search.* Our approach formulates indirect search using a finite set of relationships (FRONT-OF, NEAR, LEFT-OF, etc.) between objects. The relationships may be known apriori or, more interestingly, derived from experience with the scene. Initially objects will be represented as a (perhaps partial) local coordinate system (a circularly symmetric object might only have a Z axis and origin, for example) and a feature vector. Characterizing the occurrence of relationships as Bernoulli trials leads to a confidence interval representation of the probability of the relations holding. In turn, these probabilities can be used in a "highest impact first" search that acquires information in the order that maximally decreases expected uncertainty. The result is to derive Garvey-like strategies on the fly, with learning, and from first principles.

The third area of object search involves reasoning about obstacles and occlusion to the extent that they affect the task of finding the desired object. This research is in progress. We would like a system which can reason, for example, that since it hasn't yet seen the object, but the area under the desk has not been examined, then this area should be examined. Many issues are present in this problem. The largest is the choice of a world representation which can support this reasoning without being computationally problematic. The reasoning and world modeling must also be robust to sensor noise and marginal errors in the depth estimation process used to detect occlusions in the scene.

Wixson's work assumed a solution to the object recognition problem. Mike Swain investigated color cues for object recognition [Swain 1988a,b]. Fig. 10 shows 19 pairs of images (the originals are colored): on the left of each pair is a catalog entry,

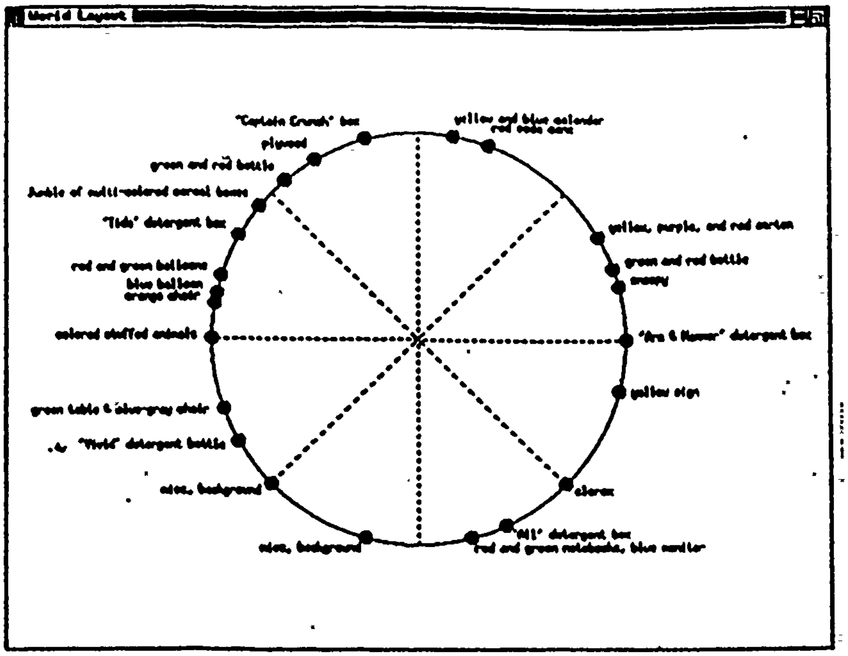


Figure 9a

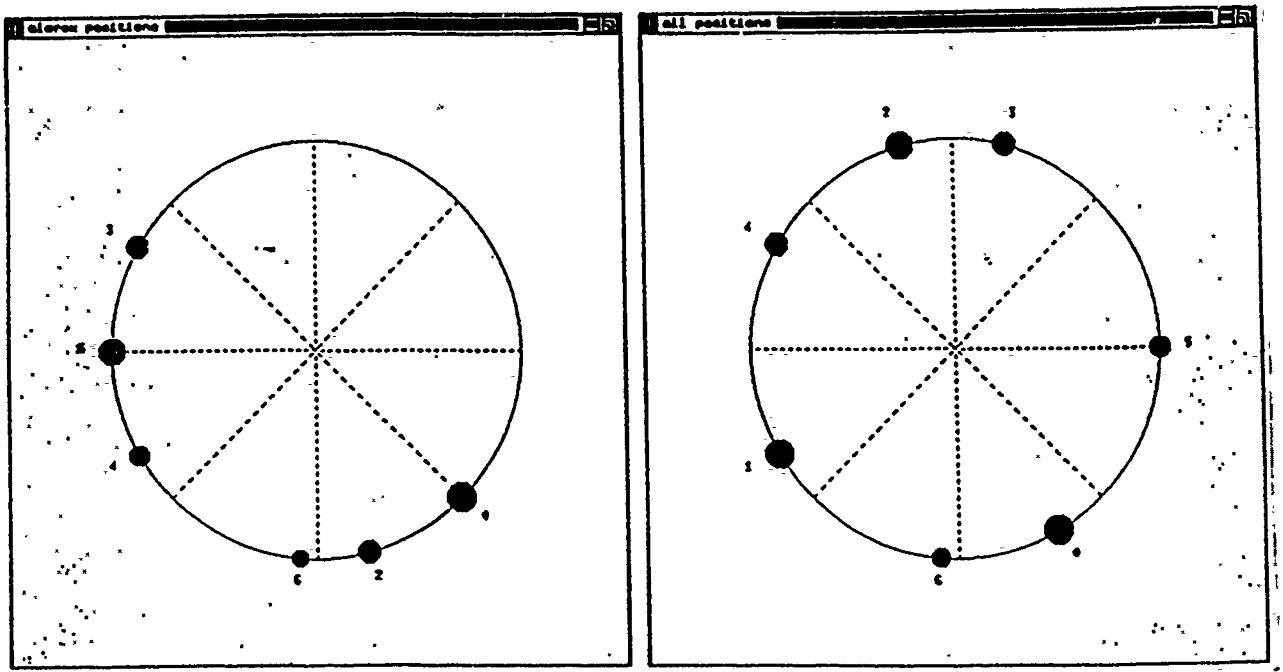


Figure 9b

Figure 9: (a) Top-view of the laboratory environment for a typical test run showing the direction (but not the distance) of each object with respect to the robot. (b) Gaze directions produced by the object search mechanism for the "Clorox" and "All" detergent boxes. Area of circle is proportional to the confidence of detection in that gaze. Numbers next to circles reflect the ordering of the confidences in decreasing order. The dashed lines in each circle are merely to provide reference points.

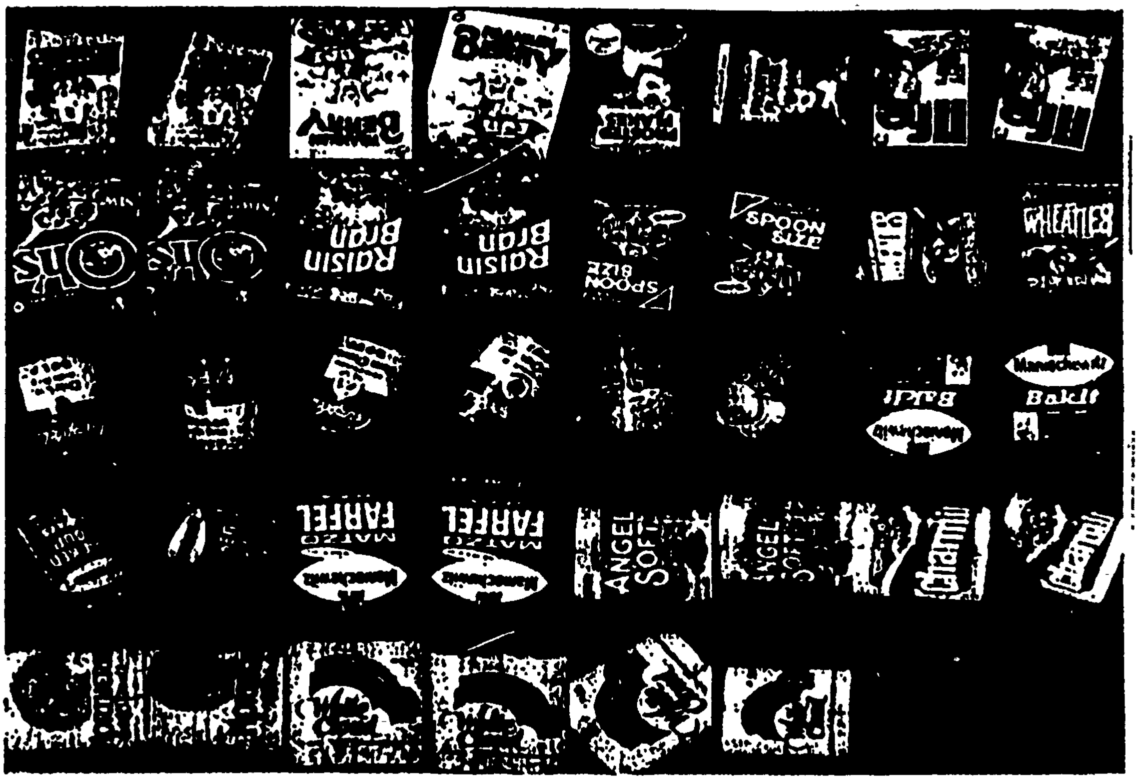


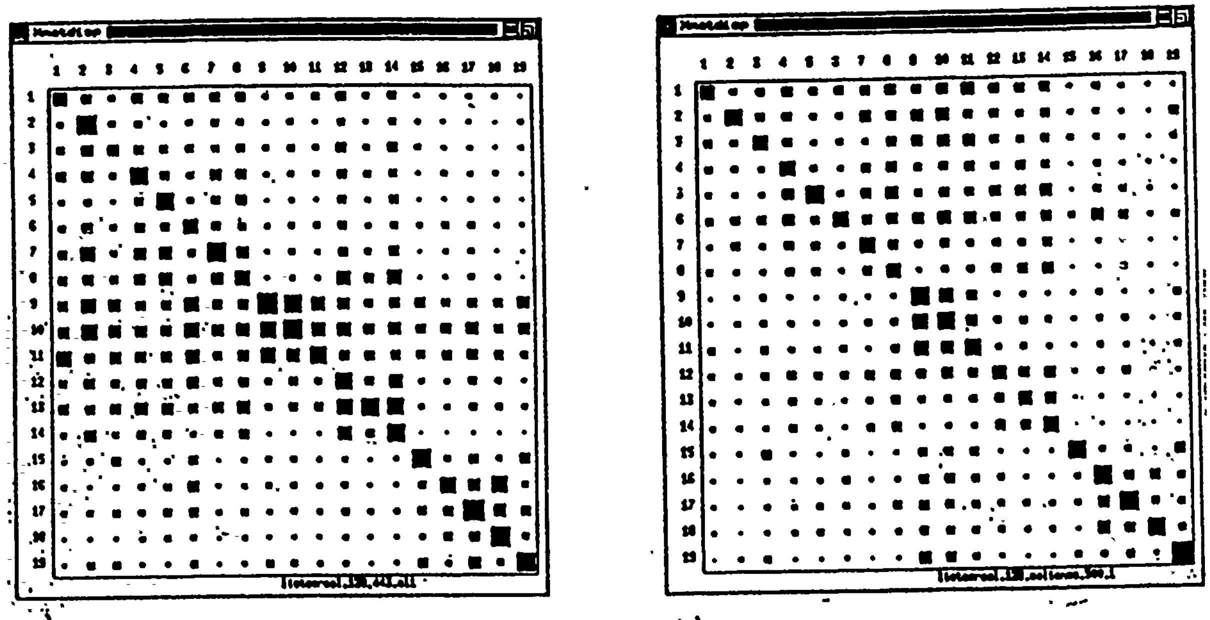
Figure 10: Black and white reproduction of color originals of (catalog, instance) image pairs.

on the right an instance from a real scene. 11 shows confusion matrices for the 19 image instances recognized from their catalog descriptions. The instance views have different viewing angles from those that generated the catalog. The basic description is a color histogram, and a saliency measure subtracts histogram features common to the ensemble, thus weighting more heavily the features that are unique to each object.

4.6 Gaze Control

In research carried out at Oxford, Chris Brown did work on Kalman filters for tracking applications (reported in the *DARPA IU Proceedings*), on projectively invariant matching of geometric structures in images (reported in the European Vision Conference), and on control of Rochester's robot head [Brown 1989b,c; 1990a,b].

The work investigated predictive mechanisms to solve problems of cooperation and delay. "Subsumption" architectures like those of Brooks and Connell find these problems troublesome since internal state representations are minimized, control interaction is usually limited to preemption, and actions are synchronized only through the outside world.



a) b)

Figure 11: Color recognition confusion matrices for pairs in previous figure (considered left to right within top to bottom.) (a) Without saliency weighting on features. In this case, the ranks of the correct choice are as follows (they should be identically 1): 2 1 4 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 . (b) With saliency weighting, the correct choices uniformly rank first.

The work developed eight camera controls and investigated their interaction. It showed that predictive techniques can overcome the catastrophic effects of delays and interactions. It made comparisons with primate gaze controls and with an open-loop approach to delay. Tracking, gaze shifts, and vergence controls used three dimensional, not retinal, coordinates. Optimal estimation techniques were used to estimate and predict the dynamic properties of the target.

The control algorithms are run in a simulation that is meant to be general and flexible, but especially to capture the relevant aspects of the Rochester Robot. Previous work with the Rochester Robot had already produced several implementations of potential basic components of a real-time gaze-control system These components included basic capabilities of target tracking, rapid gaze shifts, gaze stabilization against head motion, verging the cameras, binocular stereo, optic flow and kinetic depth calculations. These separate capabilities do not yet cooperate to accomplish tasks. The work at Oxford was partly motivated by the need to integrate several capabilities smoothly for a range of tasks useful for perception, navigation, manipulation, and in general "survival".

There are four main coordinate systems of interest in this work: LAB, HEAD, and (left and right) camera and retinal (Fig. 12). The LAB, HEAD, and camera systems are three-dimensional, right-handed and orthogonal. The retinal system is two-dimensional and orthogonal. LAB is rigidly attached to the environment in which the animate system and objects move. HEAD is rigidly attached to the head, and (for this work) has three rotational and three translational degrees of freedom. The camera systems are rigidly attached to the cameras and have independent pan and a shared tilt degree of freedom. The retinal systems represent image coordinates resulting from perspective projection of the visible world. The cameras are supported on a kinematic chain so that their principal points do not in general lie on any head rotation, pan, or tilt axis.

The simulated system controls are summarized in Table 1. Our purpose was to investigate, with some flexibility, the interactions of various forms of basic camera and head controls. The controls are not meant to model those of any biological system. Rather the goal was to build a system with sufficient functionality to exhibit many control interactions. The interaction of a subset of these controls on target tracking and acquisition tasks (the "smooth pursuit" and "saccadic" systems) was investigated and was used to illustrate the effects of different control algorithms for coping with delays.

Fig. 13. shows five of the control systems. These controls can act together (Fig. 14) to achieve different complex visual tasks such as quick target acquisition and then tracking (Fig. 15). Extending the control system to deal with delays requires kinematic simulation of the head and dynamic simulation of the outside world (Fig. 16).

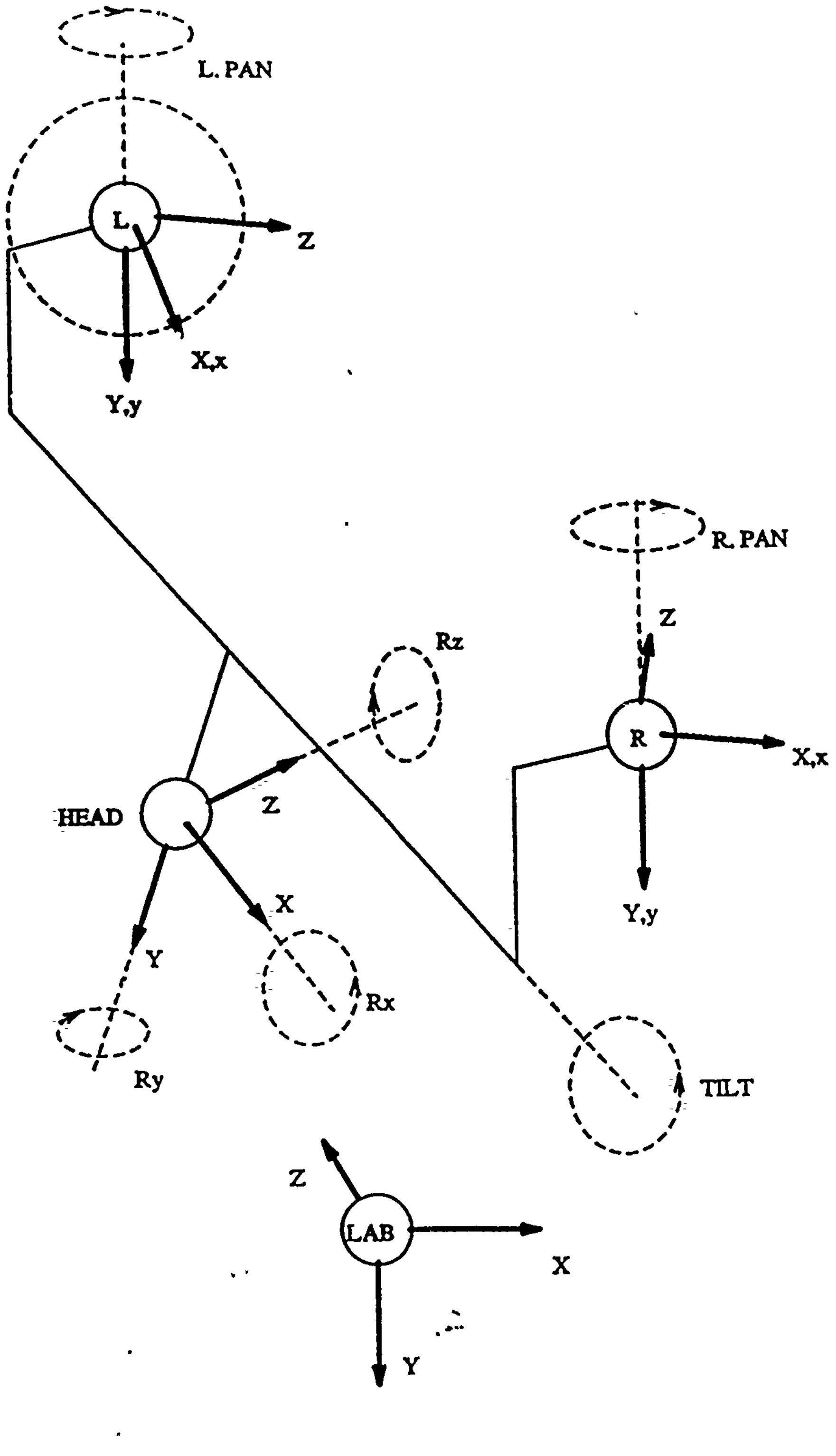
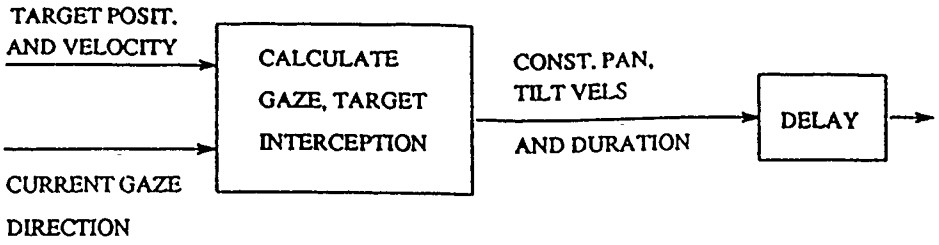
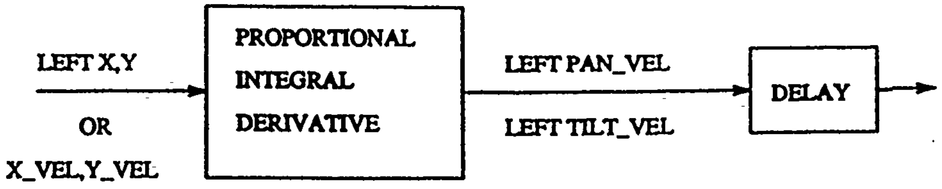


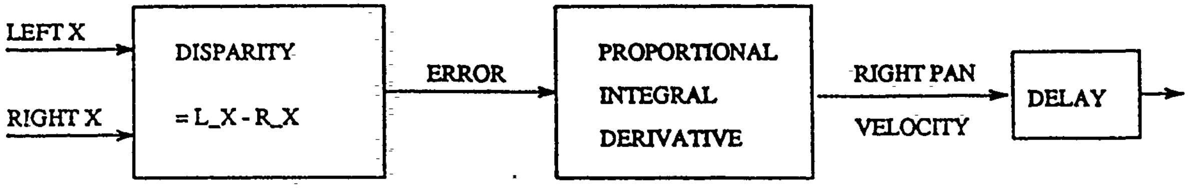
Figure 12: The coordinate systems of the simulated robot head.



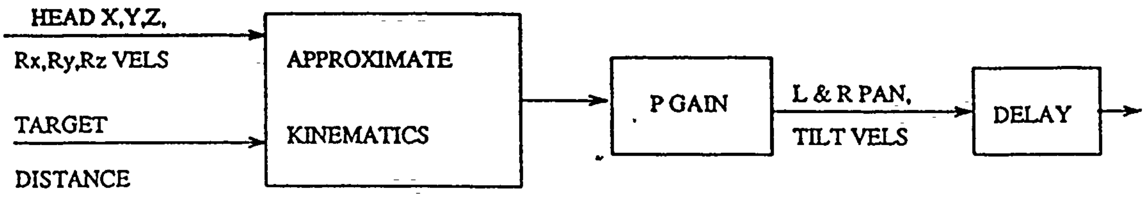
(a) RAPID GAZE SHIFT



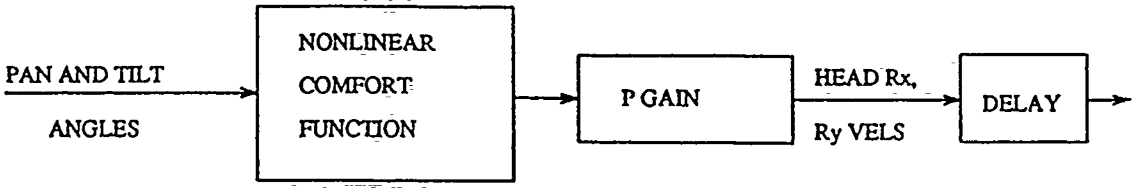
(b) TRACKING



(c) VERGENCE



(d) GAZE STABILIZATION



(e) HEAD COMPENSATION

Figure 13: Five representative head and camera controls.

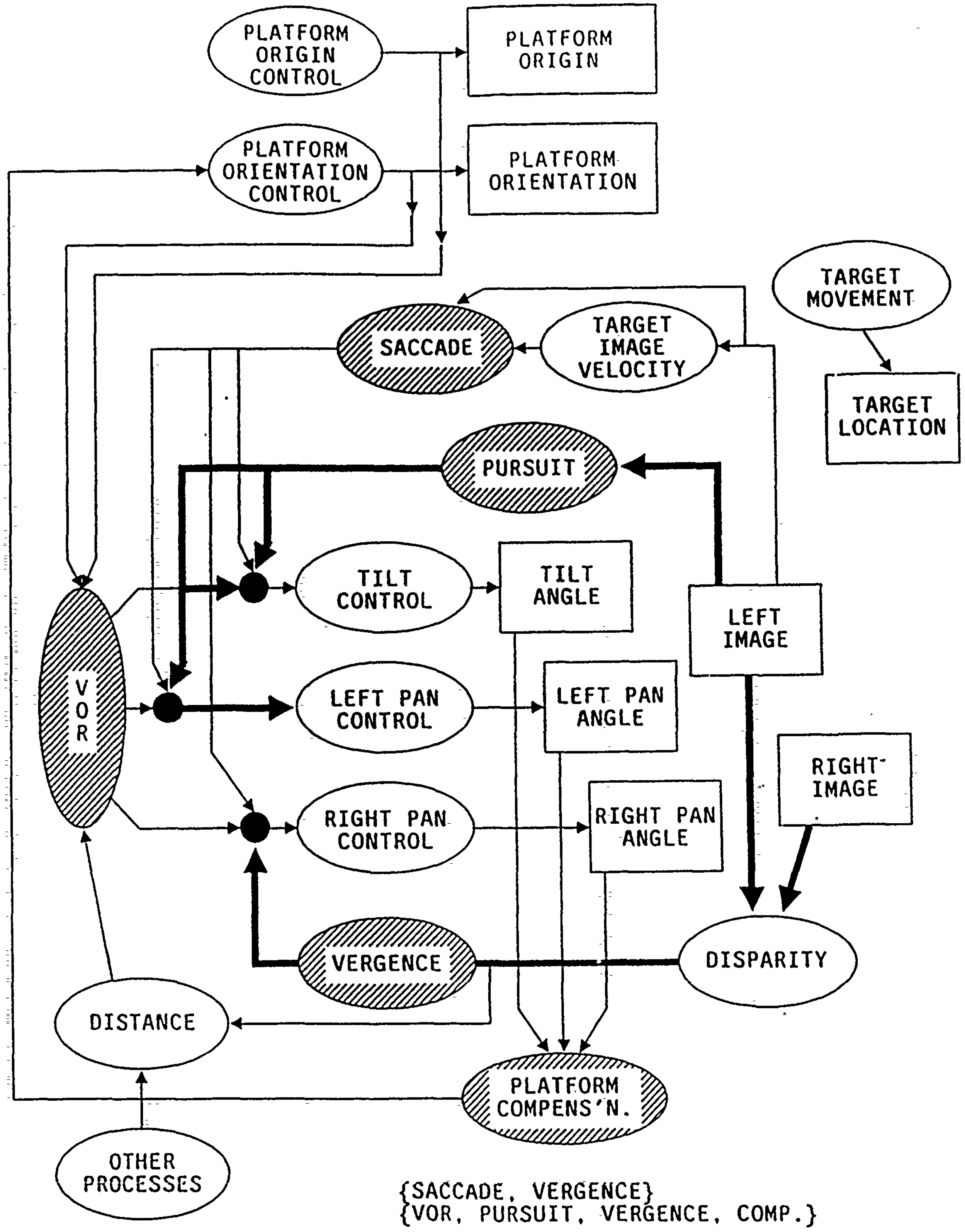


Figure 14: The interaction of the independent controls.

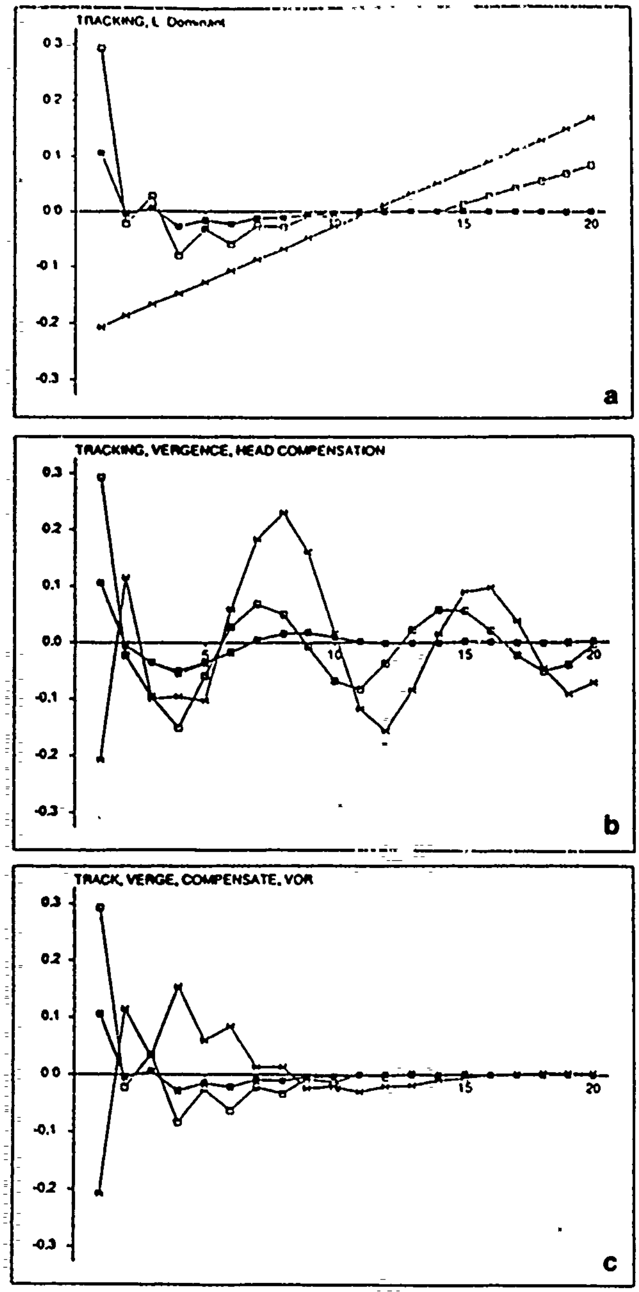


Figure 15: Increasingly effective delay-free control results from superposition of non-interacting controllers. Left and right pan and tilt angular errors in gaze direction (in radians) are plotted against time. The hollow square shows left camera pan error, the butterfly right camera pan error, and the dark square and hourglass show left and right tilt errors. (a) Tracking reflex only (one dominant eye, mechanical stops are hit); (b) adding vergence and head compensation destabilizes the system; (c) adding vestibulo-ocular (gaze stabilization) reflex stabilizes system and tracking proceeds faultlessly.

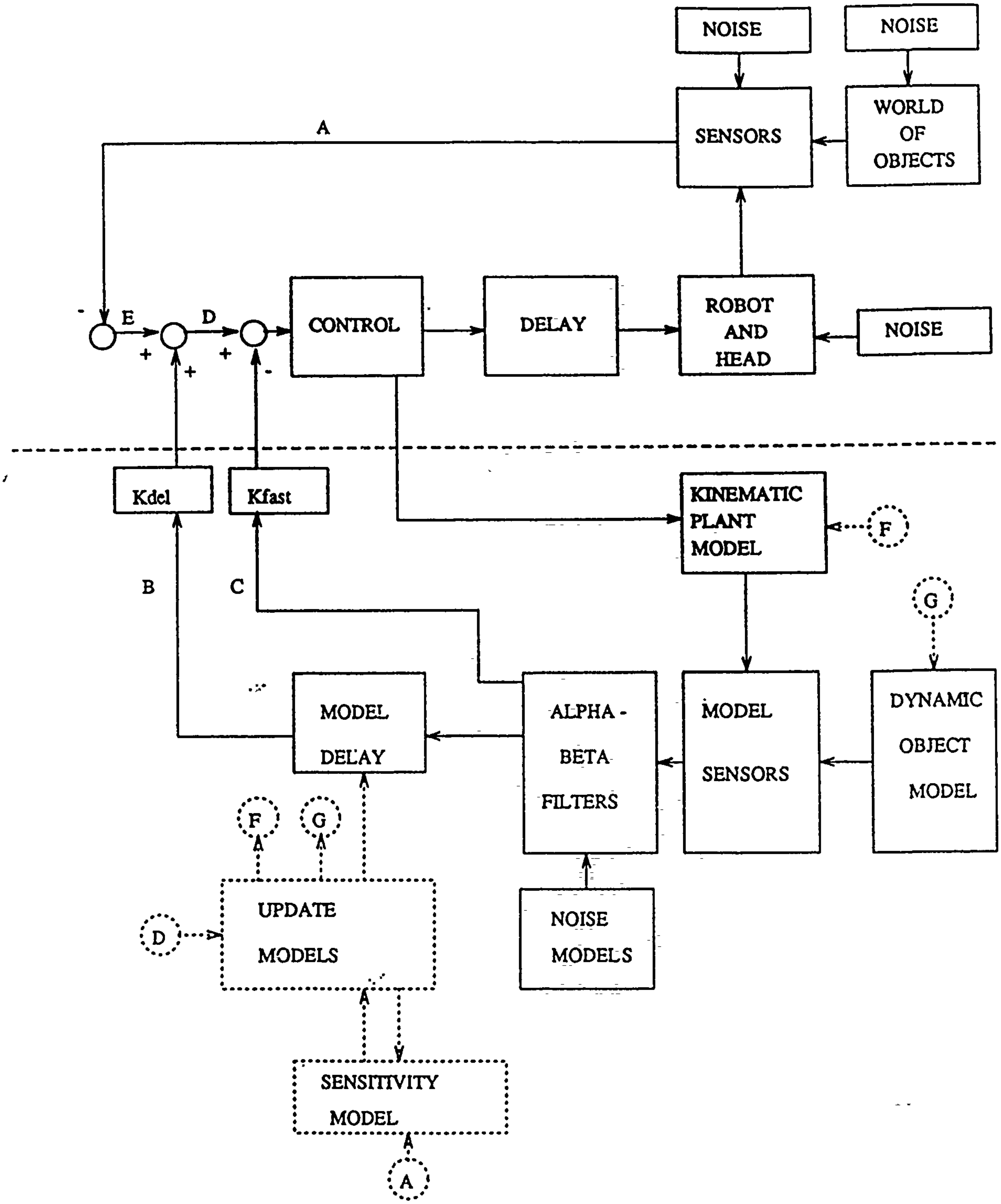


Figure 16: The extended control algorithm for delayed system.

CONTROL	**INPUT**	**ALT. INPUT**	**OUT**
EYE			
Gaze Shift	Target (x, y),$(\dot{x}, \dot{y})$	Target X,Y,Z, $\dot{X}$,$\dot{Y}$,$\dot{Z}$	L. Pan, Tilt vel.
Track	Target (x, y)	Target $(\dot{x}, \dot{y})$	L. Pan, Tilt vel.
Gaze Stabilize	Head Origin $\dot{R}x$, $\dot{R}y$, $\dot{X}$, $\dot{Y}$, $\dot{Z}$		L. Pan, Tilt vel.
Vergence	Horiz. Disparity		R. Pan vel.
Virtual Position	target (X, Y, Z)		L. Pan, Tilt vel.
HEAD			
Compensate	Eye Pans, Tilt		$(\dot{R}x, \dot{R}y)$
Fast Head Rotate	Target (X, Y, Z)		$(\dot{R}x, \dot{R}y)$
Virtual Position	Target (X, Y, Z)		$(\dot{R}x, \dot{R}y)$

Table 1: Eye and head control summary. The ALT. INPUT column shows alternate forms of input. (x, y) are image coordinates, (X, Y, Z) are world coordinates, (Rx, Ry) are head rotation angles. A design issue is whether fast gaze shifts and tracking are performed only by the "dominant eye" camera or by both cameras. Likewise vergence can affect both cameras or the non-dominant camera.

4.7 Parallel Cooperating Agents and Juggler

Our first robotics application, a balloon bouncing program called Juggler, successfully ran in November 1989 [Yamauchi 1989]. This application combines binocular camera input, a pipelined image processor, and a 6-degree-of-freedom robot arm (with a squash racquet attached) to bounce a balloon. The implementation uses a competing agent model of motor control; five processes compete with each other for access to the robot arm to position the balloon in the visual field, to position the racquet under the balloon, and to hit the balloon.

Each application process is allocated a physical processor, so scheduling is not a concern. Juggler is robust because even if processes had to share processors, failure to execute any one process during a particular time interval would have little if any affect on behavior; in the competing agent model, each application process continually broadcasts commands to the robot in competition with other processes.

Juggler was a first attempt to integrate our operating systems efforts with the development of applications. As a result of our experiences with Juggler, we are making appropriate extensions to Psyche and communications capabilities, and we have begun to experiment with user-level scheduling.

4.8 The Workbench for Active Vision Experimentation

The Workbench for Active Vision Experimentation (WAVE) has been an ongoing effort since the summer of 1988. Its purpose is to provide a uniform and general purpose platform for experimental verification of our research [Brown 1988a,b; Rimey 1990].

WAVE essentially was the first effort to "integrate everything in the Lab". The original goals were to build a system which causes the Puma robot to visually explore its environment for racquet balls randomly hanging from the Lab ceiling, and also to produce an accompanying repertory of simple modular behaviors and capabilities. In this system the Robot first moves to scan the entire Lab and locates each ball using binary image analysis and stereo vision. Next the Robot moves around a ball while keeping it centered in the field of view. A simple animate vision technique is demonstrated by computing a continuously time averaged image. The accompanying Robot movement causes the background areas in the image to be blurred while the object remains clear, thus demonstrating a simple segmentation technique. Finally the robot pokes each ball with a stick. The now moving ball is visually tracked using the eye motors on the Head (another simple animate vision idea). The overall system is more fully described in reports by Brown. A further result of this effort was a guide for other members of our group on "how to use the Rochester robot".

Last summer the WAVE platform was put to further use in a study of the problem of moving the Head to view the front of, or a characteristic view of, an object. The idea which we developed was to model vision with a parameter net model to model Head movements with a basic PID controller, and then to study differential relationships between response patterns in the parameter net and the command signals sent to the PID controller. The parameter net represented an object using a Hough transform of its silhouette. Nearness to a characteristic viewpoint was related with a distortion measure over nodes in the parameter net. The system was implemented, but performed poorly. A similar effort based on a color image approach (Wixson's work, described above) performed slightly better.

Over time WAVE has evolved into a more general platform. In anticipation of moving over to the Psyche operating system running on the Butterfly parallel computer, WAVE was converted to the g++ programming language used by Psyche and WAVE was converted to use the Zebra system for programming our DataCube MaxVideo image processing hardware. Zebra currently works with the Psyche/Butterfly system as well as the original Sun machines. WAVE itself has not yet been adapted to run under Psyche.

4.9 Modeling attentional behavior sequences with an augmented hidden Markov model

Selective attention, or the intelligent application of limited visual resources, has emerged as a basic topic for a long-range program of research we are now pursuing. The concept here is that realistically any system has to deal with limited sensing and computational resources, and that therefore we should focus our study on (selective attention) mechanisms to deal with such limited resource situations.

One approach is to map the visual attention problem onto sensor allocation problems such as where to point a camera and where to allocate processing within a single image from that camera. If we assume a spatially-variant sensor (such as one with a small, high-resolution fovea and a large, low-resolution periphery) one specific problem is to decide what sequence of eye movements to make to selectively position the fovea in the scene. One aspect of the work attacks the specific problem of modeling *foveation sequences.* In most treatments of this subject, a sequence of eye movements emerges as a result of sequential cognitive effort and image analysis, and is not explicitly represented. We decided to augment the usual paradigm with a new explicit representation of probabilistic but task-dependent attentional sequencing. Explicit sequences are something like *motor skills*; they efficiently capture the effect of much cognitive activity and *feedback-mediated behavior*, and allow it to be generated quickly with low cognitive overhead.

The explicit representation is an augmented hidden Markov model (AHMM). A simple hidden Markov model can learn an emergent behavior and re-generate it as an explicit data-oblivious sequence. An AHMM incorporates a feedback sequence to modify the generated sequence. It can therefore relearn or constantly modify its own (feedback modified) explicit behavior, thus adapting to varying conditions. Two AHMM models have been developed, the first model uses a simple external feedback loop, the second model uses internal feedback which modifies the internal parameters (probabilities) of the AHMM thus effecting the generation likelihoods directly. This work has been experimentally verified using the capabilities of WAVE and the results are encouraging [Rimey and Brown 1990].

5 Planning in a Parallel System

We have been exploring ways of forming and executing strategies that involve sequences of primitive behaviors. Actions and perception are the only realistic way to bring computerized decision-making and planning into contact with reality. This "planning" capability is necessary for systems that are to be more than reflexive [Feist 1989a,b], and which must solve problems and make decisions about what to do next [Allen and Pelavin 1986; Allen et al. 1990]. Making such decisions with uncertain information under time constraints is beyond the current state of the art, although decision-making under uncertainty, reasoning about actions through time [Allen 1989; Allen and Hayes 1987], and in general the questions of what to believe and what to do next pervade all of intelligent behavior. At Rochester, these questions are being investigated in the context of ARMTRAK [Martin et al. 1990] , a micro-world under development, based on the control of model trains, designed to integrate work in natural language, planning, vision, and robotics.

Two versions of ARMTRAK have been implemented: a simulation and a set of trains coupled to the sensors associated with the Rochester Robot. The simulation allows rapid prototyping of planners and experimentation with problems posed by different layouts. Simulations invariably involve simplifying assumptions, however, so the real trains and sensors in the vision lab allow us the rare opportunity of running a symbolic planner in the real world. The train controller has been wired so that the switchyard can be operated from outside the robot room. The vision routines are able to recognize the existence of a moving train in its field of view and are able to determine the state of a switch in its field of view. The robot also knows the locations of the switches, so it can position itself to observe them. Despite its potential, the ARMTRAK implementation is currently a demonstration of concept only. It does not have a smooth interface between the LISP world, where all the work on planning takes place, and the C environment, where the vision work is implemented. Our goal is to support LISP on our multiprocessor, and to have shared data structures linking the symbolic reasoner and the perception and action components of the system, which themselves will rely on the integrated soft and hard real-time subsystems mentioned above.

For ARMTRAK and other similar systems of the future, we would like to provide a solid substrate of visuo-motor behaviors and primitive capabilities, based on well-understood real-time technology. The user of these capabilities should not have to think about the details of their operation. Likewise, primitives for cooperation, preemption, and parallel operation of these low-level capabilities should be provided: a smooth integration of hard and soft real-time systems is an important aspect of this work.

In addition to our ARMTRAK work, our studies of learning algorithms have revealed ways of learning correct primitive sequences by trial and error or training [Whitehead and Ballard 1990; Rimey and Brown 1990]. This work suggests ways that systems can learn to adapt behaviors in complex environments and lays the groundwork for building systems that satisfice.

6 Parallel Operating Systems and the Psyche Project

The centerpiece of the CER hardware grant (on which much of the RADC research was based) was the purchase in 1985 of a 128-node BBN Butterfly Parallel Processor. Over the course of the contract this machine was used to support research in parallel programming systems, computer vision, massively parallel connectionist models, and the theory of parallel computation. CER allowed us to acquire and experiment with several generations of the Butterfly Parallel Processor from BBN-ACI. In particular, a later-generation Butterfly was obtained for operating systems research and applications. Psyche is now the major activity surrounding the Butterfly. Activity in the Psyche group involves directly or indirectly two faculty members and four to six graduate students. Psyche was running its first jobs just when the CER support terminated, and since then it has been expanding in usefulness to the user community.

One goal was to create a programming environment for MIMD (Multiple instruction stream, multiple data stream) style computers. This architecture is complementary to other styles of parallel computing such as SIMD (in which identical computations are performed in parallel to different data) and neural nets. CER allowed us to acquire and experiment with several generations of the Butterfly Parallel Processor from BBN-ACI.

The problem with MIMD computation, which admits multiple independent cooperating large processes and processors to run concurrently, is that the interactions between programs (for instance their data accessing) are extremely hard to monitor and even to repeat, given the potential for race conditions and the scheduling differences that can take place from run to run. Further, there are several competing, individually adequate models of parallel programs at this level. For instance, message-passing models and shared-memory models offer rather different user views of the computational resource. Although hardware was being built (like the BBN Butterfly Parallel Processor) to support different models of parallel computation, there was a serious lack in the current state of the art of an operating system to support several such models at once.

To improve the state of the art in programming, conceptualizing, monitoring performance, and optimizing efficiency in MIMD computation, we developed systems like PSYCHE (an operating system) and MOVIOLA (a kit of performance monitoring and debugging tools.) Altogether we also produced and exported about a dozen other less ambitious systems and libraries. The interaction of the MOVIOLA debugging and performance monitoring tools have had unexpected efficacy not just in debugging but in algorithm development.

6.1 Early Work

At the time our Butterfly was purchased it was not yet clear whether shared memory would be practical in large-scale multiprocessors. Previous architectures had been limited in size; our Butterfly and its twin at BBN were for several years the largest shared-memory multiprocessors in the world, by a large margin. Potential problems with memory and interconnect contention, the management of highly-parallel shared data structures, and the need to maximize locality of reference made our purchase a risky venture. One of the most important results of our research was to show that none of these problems is insurmountable. We used the Butterfly to obtain significant speedups (often nearly linear) on over 100 processors with a range of applications that includes various aspects of computer vision [Brown et al. 1986; Brown 1988b; Olson et al. 1987; Olson 1986b,c], connectionist network simulation [Feldman et al. 1988b], numerical algorithms [LeBlanc 1987, 1988a], computational geometry [Bukys 1986], graph theory [Costanzo et al. 1986], combinatorial search [LeBlanc et al. 1988; Scott 1989], lexical and syntactic analysis [Gafter 1987, 1988], and parallel data structure management [Mellor-Crummey 1987],

We also demonstrated, through our research in parallel programming environments and tools, that shared-memory machines are flexible enough to support efficient implementations of a wide range of programming models, with both coarse and fine-grain parallelism.

From 1984 to 1987, our systems work is best characterized as a period of experimentation, designed to evaluate the potential of large NUMA (non-uniform memory access) multiprocessors and to assess the need for software tools. In the course of this experimentation we ported three compilers to the Butterfly [Scott 1989; Olson 1986a; Crowl 1988b], developed five major and several minor library packages [Crowl 1988b; Low 1986; LeBlanc 1988b; LeBlanc and Jain 1987; Scott and Jones 1988; Olson 1986; LeBlanc and Mellor-Crummey 1986; Fowler et al. 1989], and built a parallel file system [Dibble and Scott 1989a,b; Dibble et al. 1988] and two different operating systems [LeBlanc et al 1989b; Cox and Fowler 1989]. Our work with the Lynx distributed programming language [Scott 1987] yielded important information on the inherent costs of message passing [Scott and Cox 1987] and the semantics of the parallel language/operating system interface [Scott 1986]. Experience with a C++ communication library yielded similar insights for object-oriented systems [Crowl 1988b].

A major focus of our experimentation with the Butterfly was the evaluation and comparison of multiple models of parallel computing [Brown et al. 1986; LeBlanc et al. 1988; LeBlanc 1986, 1988a]. BBN had already developed a model based on fine-grain memory sharing [LeBlanc 1986]. In addition, among the programming environments listed above, we have implemented remote procedure calls [Low 1986]; an object-oriented encapsulation of processes, memory blocks, and messages [Crowl 1988b]; a message-based library package [LeBlanc 1988b]; a shared-memory model with numerous lightweight processes [Scott and Jones 1988]; and a message-based programming language [Scott 1989]. In an intensive benchmark study conducted in 1986 [Brown et al. 1986], we implemented seven different computer vision applications on the Butterfly over the course of a three-week period. Based on the characteristics of the problems, programmers chose to use four different programming models, provided by four of our systems packages. For one of the applications, none of the existing packages provided a reasonable fit, and the awkwardness of the resulting code was a major impetus for the development of yet another package [Scott and Jones 1988].

Our principal conclusion from this experimentation was that while every programming model has applications for which it seems appropriate, no single model is appropriate for every application. Just as a general-purpose uniprocessor system must permit programs to be written in a wide variety of languages (encompassing a wide variety of models of sequential computation), we formed the belief that a general-purpose multiprocessor system must permit programs to be written under a wide variety of parallel programming models. This conviction motivated development of the Psyche operating system.

6.2 Psyche Motivation

As outlined above, our early work led to several conclusions.

1) *Large-scale shared-memory multiprocessors are practical.* We achieved significant speedups (often almost linear) using over 100 processors on a wide range of applications with many different operating systems, library packages, and languages. Shared-memory multiprocessors appear to be able to support coarse-grain parallelism just as efficiently as message-based multicomputers, while simultaneously supporting very fine-grain interactions. They provide an extremely flexible foundation for general-purpose parallel computing, and for high-level vision in particular.

2) *Programmers need multiple models of parallel computation.* Though many styles of communication and process structure can be implemented efficiently on a shared memory machine, no single model can provide optimal performance for all applications. Moreover, subjective experience indicates that conceptual clarity and ease of programming are maximized by different models for different kinds of applications. In the course of our benchmark experiments [Brown et al. 1986], seven different problems were implemented using four different programming models. One of the basic conclusions of the study was that none of the models then available was appropriate for certain graph problems; this experience led to the development of the Ant Farm library package [Scott and Jones 1988]. Large embedded applications (such as vision) may well require different programming models for different components; it therefore seemed important to be able to communicate across programming models as well.

3) *An efficient implementation of a shared name space is valuable even in the absence of uniform access time.* We found one of the primary advantages of shared memory to be its familiar computational model. A uniform addressing environment allows programs to pass pointers and data structures containing pointers without explicit translation. This uniformity of naming appears to be the primary reason why programmers choose to use BBN's Uniform System package. Even when non-uniform access times force the programmer to deal explicitly with local cacheing, shared memory continues to provide a form of global name space that supports easy copying of data from one location to another.

4) *Dynamic fine-grain sharing is important for many applications.* It is often difficult to specify at creation time which data objects will be shared and which

private, which local and which global, which long-lived and which temporary. It can be particularly difficult to specify which processes will need access to specific pieces of data, and wasteful to require processes to demonstrate access rights for data they may never use. Far preferable is a scheme in which all objects are potentially sharable and treated uniformly, with access control and other bookkeeping performed as late as possible. Such a scheme provides the user with greater latitude in program design, minimizes resource usage, and facilitates migration to maximize locality and balance workloads.

5) *Maximum performance and flexibility depend on a low-level kernel interface.* From the point of view of an individual application, the ideal operating system probably lies at one of two extremes: it either provides every facility the application needs, or else provides a flexible and efficient set of primitives from which those facilities can be built. A kernel that lies in between is likely to be both awkward and slow: awkward because it has sacrificed the flexibility of the more primitive system, slow because it has sacrificed its simplicity. Moreover a kernel with a high-level interface is unlikely to be able to provide facilities acceptable to *every* application. Low-level primitives can be much more universal. They imply the need for friendly software packages that run on top of the kernel and under user programs, but with a carefully-designed interface these can be as efficient as kernel-level code and much less difficult to change.

6) *A high-quality programming environment is essential.* Some application programmers in our department who could have exploited the parallelism offered by the Butterfly continued to use Sun workstations and VAXen. These programmers weighed the potential speedup of the Butterfly against the programming environment of their workstation and found the Butterfly wanting. Of particular importance are tools for parallel debugging. Our work with Instant Replay [LeBlanc and Mellor-Crummey 1987; Fowler et al. 1988] clearly provided an important step in the right direction. A high-quality environment for performance monitoring, called Moviola, was also created.

6.3 Psyche

Preliminary ideas for Psyche date to 1986. Design work began in earnest in 1987, and was essentially completed by the summer of 1988, when implementation began on the BBN Butterfly Plus multiprocessor. Early plans for Psyche were summarized in a 1987 technical report [Scott and LeBlanc 1987]. Rationale for the design was presented in 1988 [Scott and Marsh 1988]. Technical reports on the user/kernel interface [Scott et al. 1989a] and the memory management system [LeBlanc et al. 1989a] appeared in 1989, and were followed by workshop papers on open-systems design and the kernel implementation [Scott et al. 1989b,c]. A detailed discussion of multi-model programming appeared at the 1989 PPoPP conference.

The design of Psyche is based on the observation that access to shared memory is the fundamental mechanism for interaction between threads of control on a multiprocessor. Any other abstraction that can be provided on the machine must be built from this basic mechanism. An operating system whose kernel interface is based on direct use of shared memory will thus in some sense be universal.

The **realm** is the central abstraction provided by the Psyche kernel. Each realm includes data and code. The code constitutes a protocol for manipulating the data and for scheduling threads of control. The intent is that the data should not be accessed except by obeying the protocol. In effect, a realm is an abstract data object. Its protocol consists of operations on the data that define the nature of the abstraction. Invocation of these operations is the principal mechanism for communication between parallel threads of control.

The **thread** is the abstraction for control flow and scheduling. All threads that begin execution in the same realm reside in a single **protection domain.** That domain enjoys access to the original realm and any other realms for which access rights have been demonstrated to the kernel. Part of the layout of a thread context block is defined by the kernel, but threads themselves are created and scheduled by the user. The kernel time-slices on each processor between protection domains in which threads are active, providing upcalls at quantum boundaries and whenever else a scheduling decision is required. Context switches between threads in the same protection domain do not require kernel intervention. In addition, a standardized interface to scheduling routines allows threads of different types to block and unblock each other.

The relationship between realms and threads is somewhat unusual: the conventional notion of an anthropomorphic process has no analog in Psyche. Realms are passive objects, but their code controls all execution. Threads merely animate the code; they have no "volition" of their own.

Depending on the degree of protection desired, an invocation of a realm operation can be as fast as an ordinary procedure call or as slow as a heavyweight process switch. We call the inexpensive version an *optimized* invocation; the safer version is a *protected* invocation. In the case of a trivial protocol or truly minimal protection, Psyche also permits direct external access to the data of a realm. One can think of direct access as a mechanism for in-line expansion of realm operations. By mixing the use of protected, optimized, and in-line invocations, the programmer can obtain (and pay for) as much or as little protection as desired.

Keys and access lists are the mechanisms used to implement protection. Each realm includes an access list consisting of <key, right> pairs. Each thread maintains a list of keys. The right to invoke an operation of a realm is conferred by possession of a key for which appropriate permissions appear in the realm's access list. A key is a large uninterpreted value affording probabilistic protection. The creation and distribution of keys and the management of access lists are all under user control, enabling the implementation of many different protection policies.

If optimized (particularly in-line) invocations are to proceed quickly, they must avoid modification of memory maps. Every realm visible to a given thread must therefore occupy a different location from the point of view of that thread. In addition, if pointers are to be stored in realms, then every realm visible to multiple threads must occupy the same location from the point of view of each of those threads. In order to satisfy these two requirements, Psyche arranges for all coexistent sharable realms to occupy disjoint locations in a single, global, virtual address space. Each protection domain may have a different view of this address space, in the sense that different subsets may be marked accessible, but the virtual to physical mapping does not change.

The view of a protection domain is embodied in the hardware memory map. Execution proceeds unimpeded until an attempt is made to access something not included in the view. The resulting protection fault is fielded by the kernel, whose job it is to either (1) announce an error, (2) update the current view and restart the faulting instruction, or (3) perform an upcall into the protection domain associated with the target realm, in order to create a new thread to perform the attempted operation. In effect, Psyche uses conventional memory-management hardware as a cache for software-managed protection. Case (2) corresponds to optimized invocation. Future invocations of the same realm from the same protection domain will proceed without kernel intervention. Case (3) corresponds to protected invocation. The choice between cases is made by matching the key list of the current thread against the access list of the target realm.

For both locality and communication, the philosophy of Psyche is to provide a fundamental, low-level mechanism from which a wide variety of higher-level facilities can be built. Realms with appropriate protocol operations can be used to implement the following:

1. Pure shared memory in the style of the BBN Uniform System [Thomas 1988]. A single large collection of realms would be shared by all threads. The access protocol, in an abstract sense, would permit unrestricted reads and writes of individual memory cells.

2. Packet-switched message passing. Each message would be a separate realm. To send a message one would make the realm accessible to the receiver and inaccessible to the sender.

3. Circuit-switched message passing, in the style of Accent [Rashid and Robertson 1981] or Lynx [Scott 1987]. Each communication channel would be realized as a realm accessible to a limited number of threads, and would contain buffers manipulated by protocol operations.

4. Synchronization mechanisms such as monitors, locks, and path expressions. Each of these can be written once as a library routine that is instantiated as a realm by each abstraction that needs it.

5. Parallel data structures. Special-purpose locking could be implemented in a collection of realms scattered across the nodes of the machine, in order to reduce contention. The entry routines of the data structure as a whole might be fully parallel, able to be executed without synchronization until access is required to particular pieces of the data.

Psyche provides a low-level interface with uniform naming and an emphasis on dynamic fine-grained sharing. Through its use of data abstraction, lazy evaluation of protection, and parameterized user-level scheduling, it allows programs written under many different programming models to coexist and interact. The conventions of realm protocols, upcalls, and block and unblock routines provide a structure for communication across models that is, to the best of our knowledge, unprecedented. With appropriate permissions, user-level code can exercise full control over the physical resources of memory, processors, and devices. In effect, it should be possible under Psyche to implement almost any application for which the underlying hardware is appropriate. This, for us, constitutes the definition of "general-purpose parallel computing."

Psyche differs from existing multiprocessor operating systems in several fundamental ways.

1. It employs a uniform name (address) space for all its user programs without relying on compiler support for protection.
2. It evaluates access rights lazily, permitting the distribution of rights without kernel intervention.
3. It places the management of threads, and in fact their definition, in the hands of user-level code.
4. It minimizes the need for kernel calls in general by relying whenever possible on shared user/kernel data structures that can be examined asynchronously.
5. It provides the user with an explicit tradeoff between protection and performance by facilitating the interchange of protected and optimized invocations.

The kernel provides the foundation for a wide variety of future work in parallel systems as well as for applications (including real-time artificial intelligence). It is conceived as a lowest common denominator for a multiprocessor operating system, providing only those functions necessary to access physical resources and implement protection in higher layers. The three fundamental kernel abstractions are the segment, the address space, and the thread of control. All three are protected through capabilities. Unusual features include an inter-address-space communication mechanism based on explicit transfer of control between threads and a facility for reflecting memory protection violations upwards into user-space fault handlers.

As of November 1989 we were able to run our first real user applications. Implemented portions of Psyche include

- Low-level machine support: interrupt handlers, virtual memory (without paging), full support for inter-kernel shared memory, synchronous inter-kernel communication via remote interrupts, support for atomic hardware operations, remote source-level kernel debugging, and loading of the kernel via Ethernet.
- Core support for the Psyche user interface: realms, virtual processors, protection domains, keys and access lists, software interrupts, and protected and optimized invocation of realm operations.
- Rudimentary I/O to the console serial device, and remote file service via Ethernet.
- Minimal user-level tools: a simple shell, program loader, and name server, support for command-line argument passing, simple handlers for software interrupts, and standard I/O and kernel call libraries.

We expect our work on Psyche to evolve into many interrelated projects. We are already experimenting with novel and promising approaches to memory management, inter-node communication within the kernel and support for remote debugging. We are working to develop practical techniques to maximize locality of reference through automatic code and data migration. We expect our future efforts to include work on lightweight process structure, implementation and evaluation of communication models, and parallel language design. The latter subject is of particular interest. We have specifically avoided language dependencies in the design of the Psyche kernel. It is our intent that many languages, with widely differing process and communication models, be able to coexist and cooperate on a Psyche machine. We are interested, however, in the extent to which the Psyche philosophy itself can be embodied in a programming language.

The communications facilities of a language enjoy considerable advantages over a simple subroutine library. They can be integrated with the naming and type structure of the language. They can employ alternative syntax. They can make use of implicit context. They can produce language-level exceptions. For us the question is: to what extent can these advantages be provided without insisting on a single communication model at language-design time? We expect these questions to form the basis of future work.

The Psyche design was motivated and continues to be driven by the needs of application programs, primarily AI applications. Our experiences in the development of individual vision programs on the Butterfly provided the lessons upon which the Psyche design was based. We successfully used the active vision and robotics project as a vehicle for evaluating the Psyche design and implementation.

Our laboratory for active vision and robotics assumes a hardware configuration in which camera output is fed into a pipelined image processor and the general-purpose multiprocessor is reserved for higher-level planning and control. Initially, most of these higher-level functions were performed on a uniprocessor Sun. As the Psyche implementation became available, some of these functions were migrated onto the Buttterfly. By making this migration an explicit part of the development process we permitted early work in the systems and application domains to proceed in a semi-decoupled fashion, with neither on the other's critical path. The success of our previous efforts in operating system implementation for the Butterfly [Mellor-Crummey et al. 1987], together with the fact that Psyche construction is now well underway, suggests that the availability of the operating system is unlikely to be a problem in later phases of the project.

Research in this direction is continuing, with further hardware support provided by an NSF IIP grant. Once software has moved to the Butterfly, we expect our higher-level functions to involve hundreds of parallel threads of control. Some of these threads will share data structures. Others will interact through message passing. Some will be confine their activities to the multiprocessor. Others will interface to the image processor and the camera and robot controls. Those that share data are likely to differ in their needs for synchronization and consistency.

7 Programming Environments for Pipelined Parallel Vision: Zebra and Zed

Under the RADC contract, Rochester developed an object oriented programming interface to Datacubes MaxVideo family of image processing boards. The system is called Zebra. Zebra is not simply a packaged version of the standard Maxware calls, but rather a different style of programming for the Datacube hardware.

The basic philosophy of Zebra is two-fold. First, each board type is represented by an object class. Each physical MaxVideo board is represented by an instance of its class. Simply by declaring the board objects as variables, the boards are opened and initialized. Second, Zebra takes a microprogramming-like approach to controlling Datacube boards. The register set for each board is considered to be a microinstruction word. This instruction word completely specifies a board configuration. By sending instruction words to boards, the hardware can be completely programmed in a microprogramming-like manner.

The nature of applications code becomes largely different from that of Maxware counterparts. The configuration of MaxVideo boards is not represented in the call sequence of the application program but rather in a text file which may be changed without recompiling the application program. Thus the development process is streamlined by requiring fewer compilations.

Instruction words can be stored in and retrieved from files, allowing the sharing of standard configurations between developers. Instruction words are created an modified via an instruction word editor. One such editor "Zed" is provided with Zebra.

Zed allows a programmer to create a new instruction word or modify an existing one directly from the keyboard. This instruction word may then be saved in a file or loaded directly into a physical board for testing. This allows rapid prototyping of board configurations.

Some details of Zebra are the following.

- It is object oriented, and written in C++: It encapsulates each board as an object, created and initialized upon declaration, that can be updated and queried.
- It leads to far less complicated applications code than Maxware.
- It uses explicit human- and program- read/writeable board descriptions, which are a succinct and stable way to store, access, re-use, and share board configurations.
- It is not based on any other interface software (it does not use Maxware or the Datacube device driver, for instance).
- It already runs on two dissimilar architectures at UR (the BBN-ACI Butterfly Parallel Processor and Suns). It only assumes a memory-map operating system call and so is highly portable between host architectures.

Rochester has also developed **Zed**, which is released with **Zebra**. **Zed** has the following characteristics.

- It is an illustrative **Zebra** application.
- It provides an interactive, menu-based interface for board configuration, editing, and experimentation.
- It runs on any standard terminal, and under Suntools and X-windows.
- It allows new users to begin using Datacube hardware in minutes.

The following example Zebra program uses the P3 bus to implement a full-frame continuous transfer of image data from Digimax to a ROI-Store 512, back to Digimax, and up onto a monitor.

```
          :
    main()
    {

    // create and init the boards

1    dgBoard digimax(DG_00_BASE, DG_00_IVEC, "ZdgInit.zff");

2    rsBoard rs0(RS_00_RBASE, RS_00_MBASE, RS_M512, RS_00_IVEC,
                 "ZrsCont512.zff");

     // fire the transfer

3    rs0.fire(RS_READ);

4    rs0.fire(RS_WRITE);
    }
```

Line one declares an object of class dgBoard with the name digimax. This opens the board specified at VME address "DG_00_BASE", and initializes the board with the configuration in file "ZdgInit.zff". Line two similarly declares a roistore board object. Lines three and four are analogous to Maxware rsRFire and rsWFire respectively. Note that to change this program to do a singleshot "snapshot" transfer, the configuration file can be changed without recompiling the program. Alternatively a different configuration file can be used. Zebra and Zed are available free of charge by anonymous FTP from CS.Rochester.Edu.

8 Other Programming Libraries and Utilities for MIMD Parallelism

Several low-level communications utilities were written to support the interaction of parallel image processing with action. Communication between the embedded controller in the robot arm and controlling software on the host is via 9600 baud serial line. On top of the serial line is layered a reliable data link protocol, implemented under Unix as a tty line discipline and in the robot controller as a part of the VAL execution monitor. Above the data link layer is a protocol supporting multiple logical channels between the robot and the host. The data link software was developed and distributed by the Electrical Engineering Department at Purdue University. The logical channel software (BOTLIB) was inspired by an analogous interface developed at Purdue, but has been completely re-engineered at Rochester to provide more flexibility and speed. It provides routines to get the current robot location in terms of standard coordinates or joint angles, move the robot to a specified location in terms of standard coordinates or joint angles, set the speed of the robot, and to set the location and orientation of the tool tip. The software is organized as a C language library. The routines described above can be called from the application program.

An alternate C library (ROBOCOMM) was written by Brian Yamauchi for use in the Juggler project (see below). ROBOCOMM is much faster than the BOTLIB package since it does not use the multi-layered, reliable ISO-standard structure for communication.

Work in these areas is continuing past the contract period. Connection between the Butterfly serial ports and the robot is being explored by Mark Crovella, who is adding Psyche capabilities to manage such communications. When complete, this facility will give individual Butterfly processors the ability to communicate directly with the robot.

Under the RADC contract, Rochester developed several compilers, program libraries, systems utilities for communication, and file systems. The results at the end of the contract period span a broad range from parallel file systems through new languages for expressing parallel computation. Applications packages such as the current version of the neural net simulator [Fanty 1986, 1988; Goddard et al. 1989] and the image-processing utilities produced throughout the contract period allow speedups of up to a factor of 100 over single-workstation implementations [Olson et al 1987, Olson 1986b,c]. User interfaces to large multiprocessor computers are a difficult issue, but we have contributed to that as well [Scott and Yap 1988; Yap and Scott 1990, Olson 1986a] and we are still working to extend the range of computational models available to a user. For instance the Ant Farm project provides the basic capability to support many lightweight processes.

"An Empirical Study of Message-Passing Overhead," by M. L. Scott and A. L. Cox, appeared at the 7th International Conference on Distributed Computing Systems in Berlin, West Germany in September 1987. It reports on efforts to optimize the performance of the LYNX run-time support package, and presents a detailed breakdown of costs in the final implementation. This breakdown (1) reveals the marginal cost of various features of LYNX, (2) carries important implications for the costs of related features in other languages, and (3) sets an example for similar studies in other environments. Other work in this important effort of quantifying parallel behavior is also documented in [Floyd 1989; LeBlanc et al. 1988; LeBlanc 1988a, 1988b; Scott and Cox 1987].

The "Ant Farm" library package was used to develop applications [Scott and Jones 1989]. It supports extremely large numbers (c. 25,000) of lightweight processes in Modula-2 with location-transparent communication.

We constructed and studied the performance of a novel operating system for the Butterfly, called Elmwood. "Elmwood-An Object-Oriented Multiprocessor Operating System" appeared in Software-Practice and Experience [Mellor-Crummey et al. 1987; LeBlanc et al. 1989].

"Crowd Control: Coordinating Processes in Parallel" by T.J. LeBlanc and S. Jain appeared in the Proc. of the International Conference on Parallel Processing. This paper describes a library package for the Butterfly that can be used to create a parallel schedule for large numbers of processes. A partial order is imposed on the execution based on an arbitrary embedding of processes in a balanced binary tree [LeBlanc and Jain 1987].

Other utilities developed over the contract period include the Bridge file system for parallel I/O, by Peter Dibble [Dibble et al. 1988; Dibble and Scott 1989a,b], the Platinum and Osmium systems for automatically resolving cacheing and non-uniform reference problems in SIMD-like computations [Fowler and Cox 1988a,b; Cox and Fowler 1989]. and many other pieces of work cited in the references [Olson 1986a, Mellor-Crummey 1987; Gafter 1987, 1988; Bolosky 1989].

Characteristics of several programming utilities are compared in Table 2, which also includes some well-known programming systems for NUMA MIMD computers such as the Butterfly available commercially (Uniform System, Emerald, Linda). This extensive experience in implementing and analyzing the performance of parallel programming models has naturally led to the ideas behind the Psyche system [Scott and LeBlanc 1987; Scott et al. 1988, 1989a,b,c, 1990].

9 Programming Environments for MIMD Parallelism

A major portion of the work under the RADC contract concentrated on problems of monitoring and debugging programs for the parallel vision environment. Rochester developed many tools to help the user effectively implement parallel algorithms [e.g. LeBlanc 1989; LeBlanc et al. 1990; Mellor-Crummey 1988, 1989]. The main thrust has been the construction of parallel performance monitoring tools and experimentation with the use of these tools [e.g. Fowler and Bella 1989; Fowler et al. 1989].

One of the most serious problems in the development cycle of large-scale parallel programs is the lack of tools for debugging and performance analysis. Three issues complicate parallel program analysis. First, parallel programs can exhibit nonrepeatable behavior, limiting the effectiveness of traditional cyclic debugging techniques. Second, interactive analysis, frequently employed for sequential programs, can distort a parallel program's execution behavior beyond recognition. Third, comprehensive analysis of a parallel program's execution requires collection, management, and presentation of an enormous amount of data. Our work addressed all of these problems.

Our work has been different from other research in parallel program analysis in two key respects. First, our focus was on large-scale, shared-memory multiprocessors. Second, our approach integrated debugging and performance analysis, using a common representation of program executions.

Package	processes	scheduling	communication	synchronization	protection
Uniform System	procedure weight	concurrent; run to completion	shared memory	spin locks, atomic queues	none
Lynx	one per address space; multi-threaded	processes concurrent; threads run until blocked	RPC	implicit in communication	between processes
SMP	one per address space	concurrent, preemptable	non-blocking messages	implicit in communication	between processes
Chrysalis++	one per address space	concurrent, preemptable	shared memory, messages	events, atomic queues	between processes
Ant Farm	coroutine weight, statically located	run until blocked within a processor	shared memory	events, monitors, queues, semaphores	none
MultiLisp	coroutine weight	concurrent, preemptable	shared memory	monitors; implicit in expression evaluation	none
Platinum	multiple per address space; kernel managed	concurrent, preemptable	shared memory, messages	spin locks; implicit in communication	between address spaces
Elmwood	multiple per address space; kernel managed	concurrent, preemptable; move between objects	object invocation; shared memory within objects	implicit in invocation; semaphores and conditions within objects	between address spaces (objects)
Emerald	coroutine weight	concurrent, preemptable; move between objects	object invocation; shared memory within objects	implicit in invocation; monitors within objects	between objects (compiler enforced)
Linda	unspecified	concurrent, preemptable	shared associative store	implicit in store accesses	unspecified; often provided

Table 2: Programming systems (six developed at Rochester) for NUMA MIMD computers.

The core of our toolkit consists of facilities for recording execution histories, a common user interface for the interactive, graphical manipulation of those histories, and tools for examining and manipulating program state during replay of a previously recorded execution. These facilities form a foundation upon which we can construct more complex tools such as symbolic debuggers, execution profilers, and performance analyzers.

We have constructed a set of tools for instrumenting parallel programs on the Butterfly for performance analysis. Each process in an instrumented program records on its own "history tape" each of its interactions with shared objects including the relative timing of the operations.

An execution history is represented naturally as a directed acyclic graph (DAG) of process interactions. Nodes in the graph correspond to monitored events that took place during execution. Each event represents an operation on a shared object. Events within a process are linked by arcs denoting a temporal relation based on a local time scale. Arcs between events in different processes denote interprocess communication and synchronization.

The collection of history tapes from the individual processes can be combined to give a consistent view of the execution of the program as a whole. This view contains information useful for identifying critical paths, bottlenecks, and hot spots in the program.

An execution of a parallel program instrumented for performance monitoring generates a massive amount of data. This data is incomprehensible in its raw form so we developed an interactive graphical display and analysis program called Moviola. Moviola features a flexible user interface (graphics and LISP) and analytic tools (critical path analysis).

The "streams" package part of the NFS (Network File System) interface to the Butterfly was implemented. Mellor-Crummey produced an integrated instrumentation package that extends Instant Replay with the performance monitoring package. This uses the streams package for asynchronous transfer of "history data."

Using Moviola and the instrumentation package, we experimented with their use in the debugging and performance analysis and tuning. Mellor-Crummey's thesis demonstrated their effects in the development of parallel sorting programs [Mellor-Crummey 1989].

9.1 Performance Monitoring and Debugging

Parallel programming requires that programmers deal with new and unfamiliar abstractions, often using tools designed for sequential programs. Debugging is complicated by parallelism, and traditional cyclic techniques for debugging may not help, since many parallel programs have non-repeatable behavior. Program profilers are of little use in performance tuning, since it may be difficult to determine the impact of an individual process on overall performance, the effects of process decomposition, and the outcome of specific optimizations. Tools that report the instantaneous level of parallelism can illustrate how well the program is performing, but provide no guidance on how to improve parallelism.

For the past four years we have been developing a toolkit for debugging and performance analysis of parallel programs on large-scale shared-memory multiprocessors. Our approach is to use program replay in cyclic, post-mortem analysis Cyclic debugging assumes that experiments are interactive and repeatable, and that all relevant program behavior is observable. Unlike other work, such as Behavioral Abstraction or PIE in which monitoring software filters relevant information during execution, we save enough information to reproduce an execution for detailed analysis off-line. A distinguishing characteristic of our work is the integration of debugging and performance analysis, based on a common underlying representation of program executions.

In parallel program analysis, the focus of concern is no longer simply the internal state of a single process, but must include internal states of (potentially) many different processes and the interactions among processes. A cyclic methodology can still be used, but four issues that complicate analysis must first be addressed: (1) parallel programs often exhibit nonrepeatable behavior, (2) interactive analysis can distort a parallel program's execution, (3) analysis of large-scale parallel programs requires the collection, management, and presentation of an enormous amount of data, and (4) the execution environment, which must admit extensive parallelism, and the analysis environment, which must provide a single, comprehensive user-interface, may differ dramatically. Our research is currently devoted to addressing each of these issues.

9.2 Monitoring Parallel Programs

Monitoring parallel programs for cyclic debugging requires that essential information be extracted during execution to allow for reproducible experiments. Unfortunately, parallel programs may exhibit timing-dependent behavior due to race conditions in synchronization or programmer intervention during debugging. To allow cyclic debugging and reproducible behavior during debugging, the monitoring system must capture both program state information and relative timing information.

Several message-based debuggers have been developed that record the contents of each message sent in the system in an event log The programmer can either review the messages in the log, in an attempt to isolate errors, or the events can be used as input to replay the execution of a process in isolation. Experiments with executions can be reproduced by presenting the same messages to each process in the proper sequence.

Our approach to monitoring for shared-memory parallel programs is based on a partial order of accesses to shared objects In this approach, all interactions between processes are modeled as operations on shared objects. During program execution each process records a history of its accesses to shared objects, collecting a trace of all synchronization events that occur. The union of the individual process histories specifies a partial order of accesses to each shared object. This partial order, together with the source code and input, characterizes an execution of the parallel program. Since an execution history contains only synchronization information, it is much smaller than a record of all data exchanged between processes, making it relatively inexpensive to capture.

In addition to race conditions, other nondeterministic execution properties, such as asynchronous interrupts, can cause nonreproducible behavior. We have developed a software instruction counter to reproduce these events during program replay [Mellor-Crummey and LeBlanc 1989].

9.3 A Toolkit for Parallel Program Analysis

The information we collect during program monitoring can be used to replay a program during the debugging cycle. During replay, events can be observed at any level of detail, and controlled experiments can be performed. More important, however, is that we use program monitoring to create a representation for an execution that can be analyzed by our programmable toolkit.

The core of our toolkit consists of facilities for recording execution histories, a common user interface for the interactive, graphical manipulation of those histories, and tools for examining and manipulating program state during replay of a previously recorded execution. The user interface for the toolkit resides on the programmer's workstation and consists of two major components: an interactive, graphical browser for analyzing execution histories, and a programmable Lisp environment. The execution history browser, called Moviola, is written in C and runs under the X Windows System.

Moviola implements a graphical view of an execution based on a DAG representation of processes and communication. Moviola gathers process-local histories and combines them into a single, global execution history in which each edge represents a temporal relation between two events. In a Moviola diagram, time flows from top to bottom. Events that occur within a process are aligned vertically, forming a time-line for that process. Edges joining events in different processes reflect temporal relationships resulting from synchronization. Event placement is determined by global logical time computed from the partial order of events collected during execution. Each event is displayed as a shaded box with height proportional to the duration of the event (e.g. Fig. 17).

Moviola's user interface provides a rich set of operations to control the graphical display. Several interactive mechanisms, including independent scaling in two dimensions, zoom, and smooth panning, allow the programmer to concentrate on interesting

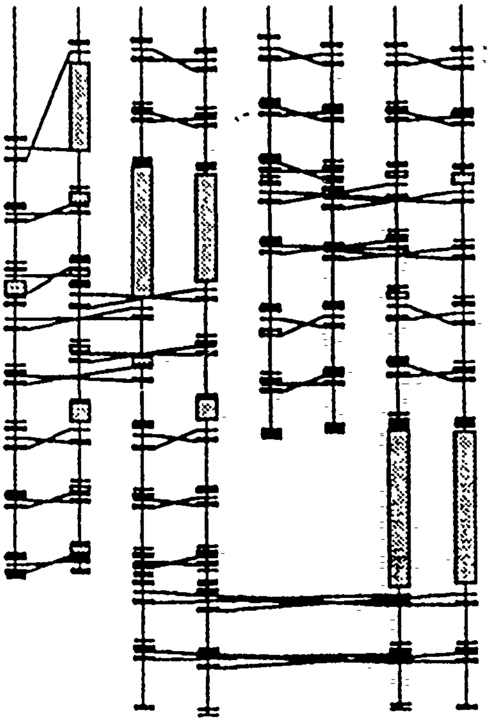


Figure 17: A Moviola diagram produced for an execution of a sort program in which the bug is clearly apparent - processors (on the right) did not hand off their work correctly, resulting in idleness and deadlock.

portions of the graph. Individual events can be selected for analysis using the mouse; the user has control over the amount and type of data displayed for selected events. The user can also control which processes are displayed and how they are displayed. By choosing to display dependencies for a subset of the shared objects, screen clutter can be reduced.

Many different analyses are possible based on this graphical view of an execution, but the sheer size of an execution history graph makes it impractical to base all analyses on manual manipulation of the graph. Extensibility and programmability are provided by running all tools under the aegis of Common Lisp. Tools can take the form of interpreted Lisp, compiled Lisp, or, like Moviola, foreign code loaded into the Lisp environment. Our programmable interface enables a user to write Lisp code to traverse the execution graph built by Moviola to gather detailed, application-specific performance statistics. The programmable interface is especially useful for performing well-defined, repetitive tasks, such as gathering the mean and standard deviation of the time it takes processes to execute parts of their computation, or how much waiting a process performs during each stage of a computation.

The programmable interface can also be used to create different views of an execution. We might want to use program animation to analyze dynamic activity over static communication channels, or application-specific views to describe the progress of a computation in terms of the program, rather than the low-level view provided by Moviola. For performance analysis, the performance graphs produced by PIE or SeeCube are much more effective than a synchronization DAG. Our current work is using the programmable interface to extend the range of views for an execution available to users, from application-specific views to detailed performance graphs (Fig. 18).

We have already constructed a mechanism for remote, source-level debugging for Psyche, in the style of the Topaz TeleDebug facility developed at DEC SRC. An interactive front end runs on a Sun workstation using the GNU gdb debugger. The debugger communicates via UDP with a multiplexor running on the Butterfly's host machine. The multiplexor in turn communicates with a low-level debugging stub (lld) that underlies the Psyche kernel.

We have successfully used this facility for kernel debugging and plan to use it as a base for user-level, multi-model debugging. Low-level debugger functions will be implemented by a combination of gdb and lld. High-level commands from the user will be translated by a model-specific interface, created as part of the programming model.

In addition, debugger stubs have been implemented to enable complex debugger queries and conditional breakpoints during execution. The toolkit has been integrated with an extended version of the gdb debugger, enabling source level debugging during replay of multiprocess programs. The Moviola graphical interface has been improved, significantly reducing the display time and increasing the functionality. The S graph-

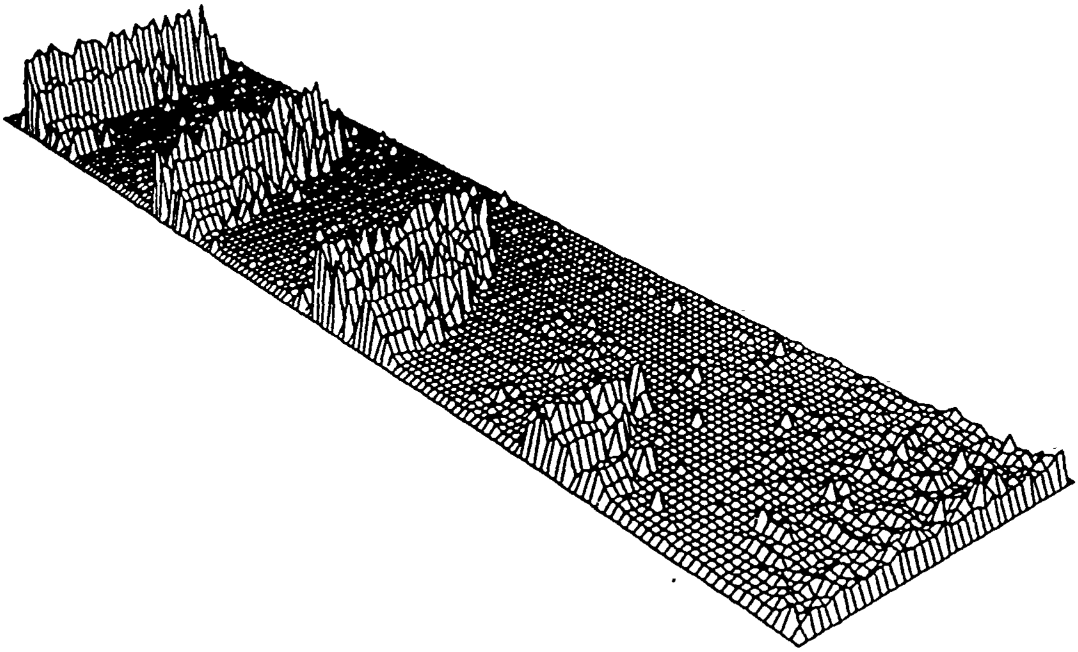


Figure 18: A perspective plot of communication time per process per row for Gaussian Elimination, as produced by the toolkit. The x-axis corresponds to the 36 processes in the computation, the y-axis corresponds to rounds of communication, one per pivot row, and the z-axis is communication time for a round. The plot shows the increase in communication time (caused by contention) as the computation progresses.

ics package has been added to the toolkit, facilitating graphical displays of performance data. LISP tools have been written for critical path analysis and for gathering and plotting performance statistics. All displays in the toolkit can be reproduced as hardcopy using Postscript format.

We are beginning to explore the relationship between program analysis, programming model (process and communication semantics), and visualization. We are investigating techniques that can be used across several parallel programming models, and a tool interface that allows a programmer to debug using the primitives provided by a particular programming model. Our goal is to (a) provide a framework that unifies our approach, as embodied in our toolkit, with the various techniques for program monitoring and visualization that have been described in the literature and (b) develop a methodology, and corresponding tools, for parallel program analysis that can be used step-by-step by programmers for the entire software development cycle, from initial debugging to performance modeling and extrapolation.

10 Technology Transfer

Under the contract Rochester developed large amounts of Butterfly applications software, the Connectionist Simulator, and the Zebra/Zed system for object-oriented register level programming. The simulator and Zebra/Zed are available by anonymous ftp or magnetic media, and hundreds of copies have been sent out worldwide.

Rochester has a substantial Industrial Affiliates Program, with industrial partners including BBN, GE, Kodak, and Xerox. In the recent past, we have had active research collaboration in the areas of vision, reasoning, and parallel programming environments with each of these affiliates. We have an annual meeting to keep affiliates abreast of our work, and to keep them aware of students and personnel here with whom they may have interests in common. Rochester students normally spend one or two summer terms working in industry, and the resulting ties with IBM, GE Research, GM Research, AT&T, and Xerox (both PARC and Webster Research Centers) are healthy and strong. These couplings are often demonstrated in observable product (the indefinite loan of the IBM 8CE computer to Fowler, the joint publications of Swain and J.L. Mundy of GE Research, etc.).

Rochester participated in the first DARPA parallel vision architecture benchmark, and the resulting applications software (as well as the other programming libraries and facilities we have developed), are disseminated through BBN. Rochester's large and well-subscribed technical reports service distributes reports to hundreds of industrial and academic sights monthly.

There is evidence that scientific papers have transferred some of the technology successfully: the Instant Replay system was implemented on Sequent computers by a group in Germany, for example. Through an international computer newsgroup the expertise on the DataCube pipelined processor is both shared and acquired. The Rochester Connectionist Simulator and the Zebra/Zed systems are available by anonymous ftp. Together they have been distributed to several hundred sites worldwide.

11 Thesis Abstracts

Several theses appeared during the contract period that were directly related to the contract. Many more were initiated during the contract period and have been completed since, or are still (1990) in process. The following are representative of earlier work under the contract.

Aloimonos, J., "Computing intrinsic images," Ph.D. Thesis and TR 198, August 1986: Several theories have been proposed in the literature for the computation of shape from shading, shape from texture, retinal motion from spatiotemporal derivatives of the image intensity function, and the like. However: (1) The employed assumptions are not present in a large subset of real images. (2) Usually the natural constraints guarantee unique answers, calling for strong additional assumptions about the world. (3) Even if physical constraints guarantee unique answers, often the resulting algorithms are not robust. This thesis shows that if several available cues are combined, then the resulting algorithms compute intrinsic parameters (shape, depth, motion, etc.) uniquely and robustly. The computational aspect of the theory envisages a cooperative highly parallel implementation, bringing in information from five different sources (shading, texture, motion, contour and stereo), to resolve ambiguities and ensure uniqueness and stability of the intrinsic parameters. The problems of shape from texture, shape from shading and motion, visual motion analysis, and shape and motion from contour are analyzed in detail.

Bandopadhay, A., "A computational study of rigid motion perception," Ph.D. Thesis and TR 221, December 1986: The interpretation of visual motion is investigated. The task of motion perception is divided into two major subtasks: (1) estimation of two-dimensional retinal motion, and (2) computation of parameters of rigid motion from retinal motion. Retinal motion estimation is performed using a point matching algorithm based on local similarity of matches and a global clustering strategy. The clustering technique unifies the notion of matching and motion segmentation and provides an insight into the complexity of the matching and segmentation process. The constraints governing the computation of the rigid motion parameters from retinal motion are investigated. The emphasis is on determining the possible ambiguities of interpretation and how to remove them. This theoretical analysis forms the basis of a set of algorithms for computing structure and three-dimensional motion parameters from retinal displacements. The algorithms are experimentally evaluated. The main difficulties facing the computation are nonlinearity and a high-dimensional search space of solutions. To alleviate these difficulties, an active tracking method is proposed. This is a closed loop system for evaluating the motion parameters. Under such a regime, it is possible to obtain form solutions for the motion parameters. This leads to a robust cooperative algorithm for motion perception requiring a minimal amount of retinal motion matching. The central theme for this research has been the evaluation of a hierarchical model for visual motion perception. To this end, the investigations revolved around three primary issues: (1) retinal motion computation from intensity images; (2) the conditions under which three-dimensional motion may be computed from retinal motion, and the efficacy of algorithms that perform such computations; (3) the active vision or closed loop approach to visual motion interpretation and what it buys us.

Chou, P. B.-L., "The theory and practice of Bayesian image labeling," Ph.D. Thesis and TR 258, August 1988: Integrating disparate sources of information has been recognized as one of the keys to the success of general purpose vision systems. Image clues such as shading, texture, stereo disparities and image flows provide uncertain, local and incomplete information about the three-dimensional scene. Spatial a priori knowledge plays the role of filling in missing information and smoothing out noise. This thesis proposes a solution to the longstanding open problem of visual integration. It reports a framework, based on Bayesian probability theory, for computing an intermediate representation of the scene from disparate sources of information. The computation is formulated as a labeling problem. Local visual observations for each image entity are reported as label likelihoods. They are combined consistently and coherently on hierarchically structured label trees with a new, computationally simple procedure. The pooled label likelihoods are fused with the a priori spatial knowledge encoded as Markov Random Fields (MRFs). The a posteriori distribution of the labelings are thus derived in a Bayesian formalism. A new inference method, Highest Confidence First (HCF) estimation, is used to infer a unique labeling from the a posteriori distribution. Unlike previous inference methods based on the MRF formalism, HCF is computationally efficient and predictable while meeting the principles of graceful degradation and least commitment. The results of the inference process are consistent with both observable evidence and a priori knowledge. The effectiveness of the approach is demonstrated with experiments on two image analysis problems: intensity edge detection and surface reconstruction. For edge detection, likelihood outputs from a set of local edge operators are integrated with a priori knowledge represented as an MRF probability distribution. For surface reconstruction, intensity information is integrated with sparse depth measurements and a priori knowledge. Coupled MRFs provide a unified treatment of surface reconstruction and segmentation, and an extension of HCF implements a solution method. Experiments using real image and depth data yield robust results. The framework can also be generalized to higher-level vision problems, as well as to other domains.

Dibble, P.C., "A Parallel Interleaved File System," Ph.D. Thesis and TR 334, March 1990: A computer system is most useful when it has well-balanced processor and I/O performance. Parallel architectures allow fast computers to be constructed from unsophisticated hardware. The usefulness of these machines is severely limited unless they are fitted with I/O subsystems that match their CPU performance. Most parallel computers have insufficient I/O performance, or use exotic hardware to force enough I/O bandwidth through a uniprocessor file system. This approach is only useful for small numbers of processors. Even a modestly parallel computer cannot be served by an ordinary file system. Only a parallel file system can scale with the processor hardware to meet the I/O demands of a parallel computer. This dissertation introduces the concept of a parallel interleaved file system. This class of file system incorporates three concepts: parallelism, interleaving, and tools. Parallelism appears as a characteristic of the file system program and in the disk hardware. The parallel file system software and hardware allows the file system to scale with the other components of a multiprocessor computer. Interleaving is the rule the file system uses to distribute data among the processors. Interleaved record distribution is the simplest and in many ways the best algorithm for allocating records to processors. Tools are application code that can enter the file system at a level that exposes the parallel structure of the files. In many cases tools decrease interprocessor communication by moving processing to the data instead of moving the data. The thesis of this dissertation is that a parallel interleaved file system will provide scalable high-performance I/O for a wide range of parallel architectures while supporting a comprehensive set of conventional file system facilities. We have confirmed our performance claims experimentally and theoretically. Our experiments show practically linear speedup to the limits of our hardware for file copy, file sort, and matrix transpose on an array of bits stored in a file. Our analysis predicts the measured results and supports a claim that the file system will easily scale to more than 128 processors with disk drives.

Floyd, R.A., "Transparency in distributed file systems," Ph.D. Thesis and TR 272, January 1989: The last few years have seen an explosion in the research and development of distributed file systems. Existing systems provide a limited degree of network transparency, with researchers generally arguing that full network transparency in unachievable. Attempts to understand and address these arguments have been limited by a lack of understanding of the range of possible solutions to transparency issues and a lack of knowledge of the ways in which file systems are used. We address these problems by: (1) designing and implementing a prototype of a highly transparent distributed file system; (2) collecting and analyzing data on file and directory reference patterns; and (3) using these data to analyze the effectiveness of our design. Our distributed file system, Roe, supports a substantially higher degree of transparency than earlier distributed file systems, and is able to do this in a heterogeneous environment. Roe appears to users to be a single, globally accessible file system providing highly available, consistent files. It provides a coherent framework for uniting techniques in the areas of naming, replication, consistency control, file and directory placement, and file and directory migration in a way that provides full network transparency. This transparency allows Roe to provide increased availability, automatic reconfiguration, effective use of resources, a simplified file system model, and important performance benefits. Our data collection and analysis work provides detailed information on short-term file reference patterns in the UNIX environment. In addition to examining the overall request behavior, we break references down by the type of file, owner of file, and type of user. We find significant differences in reference patterns between the various classes that can be used as a basis for placement and migration algorithms. Our study also provides, for the first time, information on directory reference patterns in a hierarchical file system. The results provide striking evidence of the importance of name resolution overhead in UNIX environments. Using our data collection analysis results, we examine the availability and performance of Roe. File open overhead proves to be an issue, but techniques exist for reducing its impact.

Friedberg, S.A., "Hierarchical process composition: Dynamic maintenance of structure in a distributed environment," Ph.D. Thesis and TR 294, 1988: This dissertation is a study in depth of a method, called hierarchical process composition (HPC), for organizing, developing, and maintaining large distributed programs. HPC extends the process abstraction to nested collections of processes, allowing a multiprocess program in place of any single process, and provides a rich set of structuring mechanisms for building distributed applications. The emphasis in HPC is on structural and architectural issues in distributed software systems, especially interactions involving dynamic reconfiguration, protection, and distribution. The major contributions of this work come from the detailed consideration, based on case studies, formal analysis, and a prototype implementation, of how abstraction and composition interact in unexpected ways with each other and with a distributed environment. HPC ties processes together with heterogeneous interprocess communication mechanisms, such as TCP/IP and remote procedure call. Explicit structure determines the logical connectivity between processes, masking differences in communication mechanisms. HPC supports one-to-one, parallel channel, and many-to-many (multicasting) connectivity. Efficient computation of end-to-end connectivity from the communication structure is a challenging problem, and a third-party connection facility is needed to implement dynamic reconfiguration when the logical connectivity changes. Explicit structure also supports grouping and nesting of processes. HPC uses this process structure to define meaningful protection domains. Access control is structured (and the basic HPC facilities may be extended) using the same powerful tools used to define communication patterns. HPC provides escapes from the strict hierarchy for direct communication between any two programs, enabling transparent access to global services. These escapes are carefully controlled to prevent interference and to preserve the appearance of a strict hierarchy. This work is also a rare case study in consistency control for non-trivial, highly-available services in a distributed environment. Since HPC abstraction and composition operations must be available during network partitions, basic structural constraints can be violated when separate partitions are merged. By exhaustive case analysis, all possible merge inconsistencies that could arise in HPC have been identified and it is shown how each inconsistency can be either avoided, automatically reconciled by the system, or reported to the user for application-specific reconciliation.

Loui, R.P., "Theory and computation of uncertain inference and decision," Ph.D. Thesis and TR 228, September 1987: This interdisciplinary dissertation studies uncertain inference pursuant to the purposes of artificial intelligence, while following the tradition of philosophy of science. Its major achievement is the extension and integration of work in epistemology and knowledge representation. This results in both a better system for evidential reasoning and a better system for qualitative non-monotonic reasoning. By chapter, the contributions are: a comparison of non-monotonic and inductive logic; the effective implementation of Kyburg's indeterminate probability system; an extension of that system; a proposal for decision-making with indeterminate probabilities; a system of non-monotonic reasoning motivated by the study of probabilistic reasoning; some consequences of this system; a conventionalistic foundation for decision theory and non-monotonic reasoning.

Mellor-Crummey, J., "Debugging and analysis of large-scale parallel programs," Ph.D. Thesis and TR 312, September 1989: One of the most serious problems in the development cycle of large-scale parallel programs is the lack of tools for debugging and performance analysis. Parallel programs are more difficult to analyze than their sequential counterparts for several reasons. First, race conditions in parallel programs can cause non-deterministic behavior, which reduces the effectiveness of traditional cyclic debugging techniques. Second, invasive, interactive analysis can distort a parallel program's execution beyond recognition. Finally, comprehensive analysis of a parallel program's execution requires collection, management, and presentation of an enormous amount of information. This dissertation addresses the problem of debugging and analysis of large-scale parallel programs executing on shared-memory multiprocessors. It proposes a methodology for top-down analysis of parallel program executions that replaces previous ad-hoc approaches. To support this methodology, a formal model for shared-memory communication among processes in a parallel program is developed. It is shown how synchronization traces based on this abstract model can be used to create indistinguishable executions that form the basis for debugging. This result is used to develop a practical technique for tracing parallel program executions on shared-memory parallel processors so that their executions can be repeated deterministically on demand. Next, it is shown how these traces can be augmented with additional information that increases their utility for debugging and performance analysis. The design of an integrated, extensible toolkit based on these traces is proposed. This toolkit uses execution traces to support interactive, graphics-based, top-down analysis of parallel program executions. A prototype implementation of the toolkit is described explaining how it exploits our execution tracing model to facilitate debugging and analysis. Case studies of the behavior of several versions of two parallel programs are presented to demonstrate both the utility of our execution tracing model and the leverage it provides for debugging and performance analysis.

Olson, T.J., "An architectural model of visual motion understanding," Ph.D. Thesis and TR 305, August 1989: The past few years have seen an explosion of interest in the recovery and use of visual motion information by biological and machine vision systems. In the area of computer vision, a variety of algorithms have been developed for extracting various types of motion information from images. Neuroscientists have made great strides in understanding the flow of motion information from the retina to striate and extrastriate cortex. The psychophysics community has gone a long way toward characterizing the limits and structure of human motion processing. The central claim of this thesis is that many puzzling aspects of motion perception can be understood by assuming a particular architecture for the human motion processing system. The architecture consists of three functional units or subsystems. The first or low-level subsystem computes simple mathematical properties of the visual signal. It is entirely bottom-up, and prone to error when its implicit assumptions are violated. The intermediate-level subsystem combines the low-level system's output with world knowledge, segmentation information and other inputs to construct a representation of the world in terms of primitive forms and their trajectories. It is claimed to be the substrate for long-range apparent motion. The highest level of the motion system assembles intermediate-level form and motion primitives into scenarios that can be used for prediction and for matching against stored models. This architecture is the result of joint work with Jerome Feldman and Nigel Goddard. The description of the low-level system is in accord with the standard view of early motion processing, and the details of the high-level system are being worked out by Goddard. The secondary contribution of this thesis is a detailed connectionist model of the intermediate level of the architecture. In order to compute the trajectories of primitive shapes it is necessary to design mechanisms for handling time and Gestalt grouping effects in connectionist networks. Solutions to these problems are developed and used to construct a network that interprets continuous and apparent motion stimuli in a limited domain. Simulation results show that its interpretations are in qualitative agreement with human perception.

Shastri, L., "Evidential reasoning in semantic networks: A formal theory and its parallel implementation," Ph.D. Thesis and TR 166, September 1985: This thesis describes an evidential framework for representing conceptual knowledge, wherein the principle of maximum entropy is applied to deal with uncertainty and incompleteness. It is demonstrated that the proposed framework offers a uniform treatment of inheritance and categorization, and solves an interesting class of inheritance and categorization problems, including those that involve exceptions, multiple hierarchies, and conflicting information. The proposed framework can be encoded as an interpreter-free, massively parallel (connectionist) network that can solve the inheritance and categorization problems in time proportional to the depth of the conceptual hierarchy.

Sher, D.B., "A probabilistic approach to low-level vision," Ph.D. Thesis and TR 232, October 1987: A probabilistic approach to low-level vision algorithms results in algorithms that are easy to tune for a particular application and modules that can be used for many applications. Several routines that return likelihoods can be combined into a single more robust routine. Thus it is easy to construct specialized yet robust low-level vision systems out of algorithms that calculate likelihoods. This dissertation studies algorithms that generate and use likelihoods. Probabilities derive from likelihoods using Bayes's rule. Thus vision algorithms that return likelihoods also generate probabilities. Likelihoods are used by Markov Random Field algorithms. This approach yields facet model boundary pixel detectors that return likelihoods. Experiments show that the detectors designed for the step edge model are on par with the best edge detectors reported in the literature. Algorithms are presented here that use the generalized Hough transform to calculate likelihoods for object recognition. Evidence, represented as likelihoods, from several detectors that view the same data with different models are combined here. The likelihoods that result are used to build robust detectors.

12 Bibliography

Selected RADC-Related Publications
University of Rochester Computer Science Department
1984 - 1990

Albicki, A., J. Kalinowski, A. Krasniewski, and S.-K. Yap, "Computer-aided self-test with the X Windowing System," in G. Zorbrist (Ed.). *Progress in Computer Aided VLSI Design* (Vol. 1). Norwood, NJ: Ablex Publishing Corp., 313-351, March 1989.

Allen, J.F., "Maintaining knowledge about temporal intervals," in D.S. Weld and J. deKleer (Eds.). *Readings in Qualitative Reasoning about Physical Systems*. San Mateo, CA: Morgan Kaufman Publishing Co., 1989.

Allen, J.F., "Natural language, knowledge representation and logical form," in R. Weischedel and L. Bates (Eds.). *Natural Language Processing*. To appear, 1990.

Allen, J.F., "Natural language understanding," in E. Fergenbaum and P.R. Cohen (Eds.). *The Handbook of Artificial Intelligence*. Reading, MA: Addison-Wesley, to appear, 1990.

Allen, J.F., "Two views of intention: Comments on Bratman and on Cohen and Levesque," in P.R. Cohen, J.L. Morgan, and M.E. Pollack (Eds.). *Intentions in Communication*. Cambridge, MA: Bradford Books/MIT Press, to appear, 1990.

Allen, J.F., S. Guez, L.J. Hoebel, E.A. Hinkelman, K.J. Jackson, A.I. Kyburg, and D.R. Traum, "The discourse system project," TR 317, Computer Science Dept., U. Rochester, November 1989.

Allen, J.F. and P.J. Hayes, "A common-sense theory of time," *Proc., 9th Int'l. Joint Conf. on Artificial Intelligence*, August 1985.

Allen, J.F. and P.J. Hayes, "Moments and points in an interval-based temporal logic," TR 180, Computer Science Dept., U. Rochester, December 1987; to appear, *Computational Intelligence*.

Allen, J.F., H.A. Kautz, R.N. Pelavin, and J.D. Tenenberg. *Formal Models of Reasoning about Plans*. San Mateo, CA: Morgan-Kaufman Publishing Co., to appear, 1990.

Allen, J.F. and D.J. Litman, "Discourse processing and commonsense plans," in P.R. Cohen, J.L. Morgan, and M.E. Pollack (Eds.). *Intentions in Communication*. Cambridge, MA: Bradford Books/MIT Press, to appear, 1990.

Allen, J.F. and R. Pelavin, "A formal logic of plans in temporally rich domains," *Proc. of the IEEE 74*, Special Issue on Knowledge Representation, 10, October 1986.

Allen, J.F., A. Tate, and J. Hendler (Eds.). *Readings in Planning*. San Mateo, CA: Morgan Kaufman Publishing Co., to appear, 1990.

Aloimonos, J., "Computing intrinsic images," Ph.D. Thesis and TR 198, Computer Science Dept., U. Rochester, September 1986.

Aloimonos, J., A. Bandopadhay, and P. Chou, "On the foundations of trinocular machine vision," *Technical Digest* (Topical Meeting of the Optical Society of America), April 1985.

Aloimonos, J. and C.M. Brown, "Direct processing of curvilinear sensor motion from a sequence of perspective images," *Proc., IEEE Workshop on Computer Vision Representation and Control*, 72-77, June 1984a.

Aloimonos, J. and C.M. Brown, "On the kinetic depth effect," *Biological Cybernetics 60*, 6, 445-455, 1989.

Aloimonos, J. and C.M. Brown, "Perception of structure from motion, 1) Optical flow vs. discrete displacements, II) Lower bound results," *IEEE Conf. on Computer Vision and Pattern Recognition*, June 1986.

Aloimonos, J. and C.M. Brown, "The relationship between optical flow and surface orientation," *Proc., 7th ICPR*, August 1984b.

Aloimonos, J. and C.M. Brown, "Robust computation of intrinsic images from multiple cues," in Brown, C.M. (Ed). *Advances in Computer Vision* (Vol. 1). Hillsdale, NJ: Lawrence Erlbaum Assoc., Pub., 1988.

Aloimonos, J. and P. Chou, "Detection of surface orientation from texture," *Optics News*, September 1985.

Aloimonos, J. and M.J. Swain, "Shape from patterns: Regularization," CAR-TR-283, Center for Automation Research, U. Maryland, April 1987; *Int'l. J. of Computer Vision* 2, 171-187, 1988.

Aloimonos, J. and M.J. Swain, "Shape from texture," *Proc., 9th Int'l. Joint Conf. on Artificial Intelligence*, 926-931, August 1985.

Bacchus, F., H.E. Kyburg, Jr., and M. Thalos, "Against conditionalization," TR 256, Computer Science Dept., U. Rochester, June 1988.

Bacchus, F., J.D. Tenenberg, and J.A.G.M. Koomen, "A non-reified temporal logic," submitted for journal publication, 1990.

Baldwin, D., "AI, algorithms, and hybrids for electronic design," presentation, *2nd Int'l. Conf. on the Application of Artificial Intelligence in Engineering*, August 1987.

Baldwin, D., "Applying the RASP-SL description language to a benchmark suite," extended abstract, *Workshop on High-Level Synthesis*, January 1988.

Baldwin, D., "Constraint description and extraction in RASP," *Proc., 9th Int'l. Symp. on Computer Hardware Description Languages*, Washington, D.C., June 1989.

Baldwin, D., "CONSUL: A parallel constraint language," *IEEE Software*, 62-69, July 1989.

Baldwin, D., "Describing constraints on a digital circuit's behavior," TR 222, Computer Science Dept., U. Rochester, July 1987.

Baldwin, D., "Layered design," *Proc., 1989 NSF Engineering Design Research Conference*, U. Massachusetts, Amherst, June 1989.

Baldwin, D., "Preliminary estimates of parallelism in CONSUL programs," *Proc., 22nd Hawaii Int'l. Conf. on System Sciences*, Kona, HI, January 1989.

Baldwin, D., "A status report on CONSUL," *Proc., 2nd Workshop on Programming Languages and Compilers for Parallel Programming*, U. Illinois, August 1989.

Baldwin, D., "Why we can't program multiprocessors the way we're trying to do it now," TR 224, Computer Science Dept., U. Rochester, August 1987.

Baldwin, D. and C.A. Quiroz, "Parallel programming and the CONSUL language," *Proc., Int'l. Conf. on Parallel Processing*, St. Charles, IL, August 1987.

Ballard, D.H., "Animate vision," TR 329, Computer Science Dept., U. Rochester, February 1990; to appear, *AI Journal*, 1990a.

Ballard, D.H., "Animate vision uses object-centered reference frames," *Int'l. Symp. on Neural Networks for Sensory and Motor Systems* (NSMS), Düsseldorf, West Germany, March 1990b.

Ballard, D.H., "Eye movements and spatial cognition," TR 218, Computer Science Dept., U. Rochester, November 1987; presented, *AAAI Spring Symposium Series on Physical and Biological Approaches to Computational Vision*, March 1988; to appear in forthcoming book (*Proc., 1988 Workshop on Exploratory Vision: The Active Eye*).

Ballard, D.H., "Reference frames for animate vision," Best Paper Prize, *Proc., Int'l. Joint Conf. on Artificial Intelligence*, Detroit, MI, August 1989; *2nd Int'l. Congress of Neuroethology*, Berlin, September 1989.

Ballard, D.H., "Task frames in robot manipulation," *Proc., Nat'l. Conf. on Artificial Intelligence*, August 1984.

Ballard, D.H., "Task frames in visuo-motor coordination," *Proc., 3rd IEEE Workshop on Computer Vision: Representation and Control*, October 1985.

Ballard, D.H., C.M. Brown, D.J. Coombs, and B.D. Marsh, "Eye movements and computer vision," *1987-88 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, October 1987.

Ballard, D.H., R.C. Nelson, and B. Yamauchi, "Animate vision," invited article, *Optics News 15*, 5 (Special Issue on Image Understanding and Machine Vision), May 1989; *1989-90 Computer Science/Engineering Research Review*, Computer Science Dept., U. Rochester, December 1989.

Ballard, D.H. and A. Ozcandarli, "Eye fixation and early vision: Kinetic depth," *Proc., 2nd IEEE Int'l. Conf. on Computer Vision*, December 1988.

Bandopadhay, A., "A computational study of rigid motion perception," Ph.D. Thesis and TR 221, Computer Science Dept., U. Rochester, December 1986.

Bandopadhay, A. and D.H. Ballard, "Active navigation: Egomotion perception by the tracking observer," to appear, *Computational Intelligence*, 1990.

Bashir, N., M. Crovella, G. DeTitta, H. Hauptman, J. Horvath, H. King, D. Langs, R. Miller, T. Sabin, P. Thuman, and D. Velmurugan, "Parallel solutions to the phase problem in x-ray crystallography," submitted for conference publication, 1990.

Basu, A. and C.M. Brown, "Algorithms and hardware for efficient image smoothing," *Computer Vision, Graphics, and Image Processing 40*, 2, 131-146, November 1987.

Bolosky, W.J., R.P. Fitzgerald, and M.L. Scott, "Simple but effective techniques for NUMA memory management," *Proc., 12th ACM Symp. on Operating System Principles* (SOSP), 19-31, Litchfield Park, AZ, December 1989.

Brown, C.M. (Ed). *Advances in Computer Vision* (Vols. I and II). Hillsdale, NJ: Lawrence Erlbaum Assoc., Pub., 1988a.

Brown, C.M., "Computer vision and natural constraints," *Science 224*, 4655, 1299-1305, June 1984a.

Brown, C.M., "Gaze behaviors for robotics," invited paper, *Proc., NATO-ACI Symp. on Active Perception and Robot Vision*, Maratea, Italy, July 1989a.

Brown, C.M., "Gaze controls cooperating through prediction," *Image and Vision Computing 8* (Special Edition), 1, 10-17, February 1990a.

Brown, C.M., "Gaze controls with interactions and delays," TR 278, Computer Science Dept., U. Rochester, March 1989; Report OUEL 1770/89, Robotics Research Group, Dept. of Engineering Science, U. Oxford, March 1989; *Proc., DARPA Image Understanding Workshop*, Palo Alto, CA, May 1989; to appear, *IEEE Trans. Systems, Man, and Cybernetics 20*, 2, March 1990b.

Brown, C.M., "Parallel vision with the Butterfly® computer," invited paper, *Proc., Third Int'l. Conf. on Supercomputing*, May 1988b.

Brown, C.M., "Peak-finding with limited hierarchical memory," *Proc., 7th Int'l. Conf. on Pattern Recognition*, August 1984b.

Brown, C.M., "Prediction and cooperation in gaze control," to appear, *Biological Cybernetics*, 1990.

Brown, C.M., "Prediction in gaze and saccade control," TR 295, Computer Science Dept., U. Rochester, May 1989; Report OUEL 1771/89, Robotics Research Group, Dept. of Engineering Science, U. Oxford, May 1989b.

Brown, C.M., "Predictive gaze control," *Proc., 5th Alvey Vision Conf.*, U. Reading, England, September 1989c.

Brown, C.M., "Progress in image understanding at the University of Rochester," *Proc., DARPA Image Understanding Workshop*, 73-77, April 1988c.

Brown, C.M., "Some computational properties of rotation representations," TR 303 (revised), Computer Science Dept., U. Rochester, August 1989d.

Brown, C.M., "A space-efficient Hough transform implementation for object location," in E. Wegman (Ed). *Statistical Image Processing and Graphics.* Marcel-Dekker, 1987.

Brown, C.M., "Three-dimensional dynamic and kinematic prediction in gaze control," *Proc., IEEE Workshop on Interpretation of 3D Scenes*, Austin, TX, November 1989e.

Brown, C.M., et al., "Three-dimensional vision techniques for autonomous vehicles," in R.C. Jain and A.K. Jain (Eds.). *Analysis and Interpretation of Range Images.* New York: Springer-Verlag, 1989a.

Brown, C.M., J. Aloimonos, M.J. Swain, P.B. Chou, and A. Basu, "Texture, contour, shape, and motion," *Pattern Recognition Letters 5*, 2, 151-168, 1987.

Brown, C.M. and D.H. Ballard, "Vision," *Byte 10* (Special Issue on AI), 4, 245-261, April 1985.

Brown, C.M. (Ed.), with D.H. Ballard, T.G. Becker, R.F. Gans, N.G. Martin, T.J. Olson, R.D. Potter, R.D. Rimey, D.G. Tilley, and S.D. Whitehead, "The Rochester robot," TR 257, Computer Science Dept., U. Rochester, August 1988.

Brown, C.M., H. Durrant-Whyte, J. Leonard, B. Rao, and B. Steer, "Centralized and decentralized Kalman filter techniques for tracking, navigation, and control," TR 277, Computer Science Dept., U. Rochester, May 1989 (revised); also appeared as "Kalman filter algorithms, applications, and utilities," Report OUEL 1765/89, Robotics Research Group, Dept. of Engineering Science, U. Oxford, May 1989; *Proc., DARPA Image Understanding Workshop*, Palo Alto, CA, May 1989b.

Brown, C.M., C.S. Ellis, J.A. Feldman, S.A. Friedberg, and T.J. LeBlanc, "Artificial intelligence research on the Butterfly Multiprocessor," *Proc., Nat'l Academy of Sciences Workshop on AI and Distributed Problem Solving*, Washington, DC, May 1985.

Brown, C.M., C.S. Ellis, J.A. Feldman, T.J. LeBlanc, and G.L. Peterson, "Research with the Butterfly Multicomputer," *1984-85 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, October 1984.

Brown, C.M., R.J. Fowler, T.J. LeBlanc, M.L. Scott, M. Srinivas, L. Bukys, J. Costanzo, L. Crowl, P. Dibble, N. Gafter, B. Marsh, T. Olson, and L. Sanchis, "DARPA parallel architecture benchmark study," BPR 13, Computer Science Dept., U. Rochester, October 1986.

Brown, C.M. and R.C. Nelson, "Image understanding at the University of Rochester," *Proc., DARPA Image Understanding Workshop*, Palo Alto, CA, May 1989.

Brown, C.M. and R.D. Rimey, "Coordinates, conversions, and kinematics for the Rochester Robotics Lab," TR 259, Computer Science Dept., U. Rochester, August 1988.

Bukys, L., "Connected component labeling and border following on the BBN Butterfly parallel proceessor," BPR 11, Computer Science Dept., U. Rochester, October 1986.

Chou, P.B., "The theory and practice of Bayesian image labeling," TR 258 and Ph.D. Thesis, Computer Science Dept., U. Rochester, August 1988.

Chou, P.B. and C.M. Brown, "Multimodal reconstruction and segmentation with Markov Random Fields and HCF optimization," *Proc., DARPA Image Understanding Workshop*, 214-221, April 1988.

Chou, P.B. and C.M. Brown, "Multimodal segmentation using Markov random fields," *Proc., DARPA Image Understanding Workshop*, 663-670, February 1987a.

Chou, P.B. and C.M. Brown, "Probabilistic information fusion for multi-modal image segmentation," *Proc., Int'l. Joint Conf. on Artificial Intelligence*, Milan, Italy, August 1987b.

Chou, P.B. and C.M. Brown, "The theory and practice of Bayesian image labeling," IBM Research Report RC 15460, T.J. Watson Research Center, February 1990; also to appear as "Multimodal reconstruction and segmentation with Markov Random Fields and HCF optimization," to appear, *Int'l. J. Computer Vision*, 1990.

Chou, P.B. and R. Raman, "On relaxation algorithms based on Markov random fields," TR 212, Computer Science Dept., U. Rochester, July 1987.

Chou, P.B., C.M. Brown, and R. Raman, "A confidence-based approach to the labeling problem," *Proc., IEEE Workshop on Computer Vision*, November 1987.

Coombs, D.J., "Tracking objects with eye movements," Technical Report (*Proc., 4th Annual U. Buffalo Graduate Conf. on Computer Science*), Computer Science Dept., SUNY Buffalo, March 1989; *Proc., Optical Society of America Topical Meeting on Image Understanding and Machine Vision*, N. Falmouth, MA, June 1989.

Coombs, D.J. and B.D. Marsh, "ROVER: A prototype active vision system," TR 219, Computer Science Dept., U. Rochester, August 1987.

Cooper, P.R., "Parallel object recognition from structure (The Tinkertoy Project)," Ph.D. thesis and TR 301, Computer Science Dept., U. Rochester, July 1989.

Cooper, P.R., "Structure recognition by connectionist relaxation: formal analysis," *Proc., DARPA Image Understanding Workshop*, 981-993, April 1988; *Proc., 1988 Canadian Artificial Intelligence Conference*, June 1988.

Cooper, P.R. and M.J. Swain, "Domain dependence in parallel constraint satisfaction," *Proc., Int'l. Joint Conf. on Artificial Intelligence*, Detroit, MI, August 1989.

Cooper, P.R. and M.J Swain, "Parallelism and domain dependence in constraint satisfaction," TR 255, Computer Science Dept., U. Rochester, December 1988; submitted for publication.

Costanzo, J., L. Crowl, L. Sanchis, and M. Srinivas, "Subgraph isomorphism on the BBN Butterfly multiprocessor," BPR 14, Computer Science Dept., U. Rochester, October 1986.

Cox, A.L. and R.J. Fowler, "The implementation of a coherent memory abstraction on a NUMA multiprocessor: Experiences with PLATINUM (revised)," TR 263, Computer Science Dept., U. Rochester, May 1989; *Proc., Symp. on Operating Systems Principles*, 32-44, Litchfield Park, AZ, December 1989.

Crowl, L.A., "Chrysalis++," BPR 15, Computer Science Dept., U. Rochester, December 1986.

Crowl, L.A., "Concurrent data structures and actor programming under the Matroshka model," *Proc., ACM SIGPLAN Workshop on Concurrent Object-Based Programming*, 79-80, September 1988; *ACM SIGPLAN Notices 24*, 4, 79-80, April 1989a.

Crowl, L.A., "An interface between object-oriented systems," TR 211, Computer Science Dept., U. Rochester, April 1987.

Crowl, L.A., "A model for parallel programming," in TR 209 (Proc., 1988 Open House), Computer Science Dept., U. Rochester, May 1988a.

Crowl, L.A., "Shared memory multiprocessors and sequential programming languages: A case study," *Proc., 21st Hawaii Int'l. Conf. on System Sciences,* 103-108, January 1988b.

Crowl, L.A., "A uniform object model for parallel programming," *Proc., ACM SIGPLAN Workshop on Concurrent Object-Based Programming,* 25-27, September 1988; *ACM SIGPLAN Notices 24,* 4, 25-27, April 1989b.

Dibble, P.C., "Attacking the I/O bottleneck with parallelism," in TR 209 (Proc., 1988 Open House), Computer Science Dept., U. Rochester, May 1988a.

Dibble, P.C., "CD-RTOS," in Philips International (Eds). *The CD-I Designer's Guide.* Hightstown, NJ: McGraw-Hill, 1988b.

Dibble, P.C. *OS-9 INSIGHTS: An Advanced Programmers Guide to OS-9 / 68000.* Des Moines, IA: Microware Systems Corporation, 1988c.

Dibble, P.C. *OS-9 Insights Ein Programmierhandbuch für OS-9/68000.* Translated by Karl-Heinz Porombka. Heidelberg: Hüthing Buch Verlag Heidelberg, 1989.

Dibble, P.C., "OS-9/mp," invited paper, *1987 European OS-9 Convention,* September 1987.

Dibble, P.C., "A parallel interleaved file system," Ph.D. Thesis and TR 334, Computer Science Dept., U. Rochester, March 1990.

Dibble, P.C. and M.L. Scott, "Beyond striping: The bridge multiprocessor file system," *Computer Architecture News 17,* 5, 32-39, September 1989a.

Dibble, P.C. and M.L. Scott, "External sorting on a parallel interleaved file system," *1989-90 Computer Science/Engineering Research Review,* Computer Science Dept., U. Rochester, December 1989b.

Dibble, P.C., M.L. Scott, and C.S. Ellis, "Bridge: A high-performance file system for parallel processors," *Proc, 8th Int'l. Conf. on Distributed Computing Systems,* 154-161, June 1988.

Fanty, M, "A connectionist simulator for the Butterfly," TR 164, Computer Science Dept., U. Rochester, January 1986.

Fanty, M.A., "Learning in structured connectionist networks," Ph.D. Thesis, Computer Science Dept., U. Rochester, February 1988; TR 252, Computer Science Dept., U. Rochester, April 1988.

Feist, S.E., "Integrating symbolic planning and reactive execution," Thesis Proposal, Computer Science Dept., U. Rochester, August 1989a.

Feist, S., "Moving towards reactivity in planning systems," Internal Report, Computer Science Dept., U. Rochester, January 1989b.

Feldman, J.A., "Connectionist representation of concepts," in D. Waltz and J.A. Feldman (Eds). *Connectionist Models and their Applications.* Norwood, NJ: Ablex Publishing Company, 1987.

Feldman, J.A., "Time, space and form in vision," TR 244, Computer Science Dept., U. Rochester, November 1988.

Feldman, J.A. and D.H. Ballard, "Connectionist models and their properties," in D. Waltz and J.A. Feldman (Eds). *Connectionist Models and their Applications.* Norwood, NJ: Ablex Publishing Company, 13-62, 1988.

Feldman, J.A., D.H. Ballard, C.M. Brown, and G.S. Dell, "Rochester connectionist papers: 1979-1985," TR 172, Computer Science Dept., U. Rochester, December 1985.

Feldman, J.A., M.A. Fanty, and N. Goddard, "Computing with structured neural networks," *IEEE Computer 21*, 3, 91-103, 1988a.

Feldman, J.A., M.A. Fanty, N. Goddard, and K.J. Lynne, "Computing with structured connectionist networks," invited paper, *Commun. ACM 31*, 2, 170-187, February 1988b.

Finkel, R.A., M.L. Scott, Y. Artsy, and H.-Y. Chang, "Experience with Charlotte: Simplicity and function in a distributed operating system," *IEEE Trans. on Software Engineering* (Special Issue on Design Principles for Experimental Distributed Systems), June 1989.

Floyd, R.A., "Transparency in distributed file systems," Ph.D. Thesis and TR 272, Computer Science Dept., U. Rochester, January 1989.

Fowler, R.J. and I. Bella, "The programmer's guide to Moviola: An interactive execution history browser," TR 269, Computer Science Dept., U. Rochester, February 1989.

Fowler, R.J. and A.L. Cox, "An overview of PLATINUM, a Platform for Investigating Non-Uniform Memory (preliminary version)," TR 262, Computer Science Dept., U. Rochester, November 1988a.

Fowler, R.J. and A.L. Cox, "PLATINUM: A platform for investigating non-uniform memory," Internal Report, Computer Science Dept., U. Rochester, December 1988b.

Fowler, R.J., T.J. LeBlanc, and J.M. Mellor-Crummey, "An integrated approach to parallel program debugging and performance analysis on large-scale multiprocessors," *Proc., ACM Workshop on Parallel and Distributed Debugging*, 163-173, May 1988; *SIGPLAN Notices 24*, 1, 163-173, January 1989.

Friedberg, S.A., "Hierarchical process composition," Ph.D. Thesis and TR 294, Computer Science Dept., U. Rochester, October 1988.

Friedberg, S.A., "Transparent reconfiguration requires a third party connect," TR 220 (revised), Computer Science Dept., U. Rochester, November 1987.

Friedberg, S.A. and C.M. Brown, "Finding axes of skewed symmetry," *Proc., 7th Int'l. Conf. on Pattern Recognition*, Montreal, August 1984.

Friedberg, S.A. and G.L. Peterson, "An efficient solution to the mutual exclusion problems using weak semaphores," *Information Processing Letters 25*, 5, 343-347, 10 July 1987.

Gafter, N.M., "Algorithms and data structures for parallel incremental parsing," *Proc., Int'l. Conf. on Parallel Processing*, August 1987.

Gafter, N.M., "List productions in a parallel incremental parser," in TR 209 (Proc., 1988 Open House), Computer Science Dept., U. Rochester, May 1988.

Goddard, N.H., "The interpretation of visual motion: Recognizing moving light displays," *IEEE Workshop on Visual Motion*, Irvine, CA, April 1989.

Goddard, N.H., "Recognizing animal motion," *Proc., DARPA Image Understanding Workshop*, 938-944, April 1988a.

Goddard, N.H., "Representation and recognition of biological motion," *Proc., Cognitive Science Conf.*, August 1988b.

Goddard, N.H., "Representation and recognition of event sequences," *Proc., Carnegie Mellon University Connectionist Models Summer School* (June 1988), December 1988c.

Goddard, N.H., K.J. Lynne, and T. Mintz, "Rochester connectionist simulator," TR 233, Computer Science Dept., U. Rochester, revised March 1988; revised April 1989.

Hartman, L.B., "Decision theory and the cost of planning," forthcoming Ph.D. Thesis, Computer Science Dept., U. Rochester, to appear, 1990.

Hinkelman, E.A., "Abductive speech act recognition," to appear, *Proc., AAAI Spring Symp.*, Stanford, CA, March 1990.

Hollbach, S.C., "Direct inferences in a connectionist knowledge structure," in TR 209 (Proc., 1988 Open House), Computer Science Dept., U. Rochester, May 1988.

Kautz, H., "A formal theory of plan recognition," Ph.D. Thesis and TR 215, Computer Science Dept., U. Rochester, May 1987.

Kautz, H. and J.F. Allen, "Generalized plan recognition," *Proc., Nat'l. Conf. on Artificial Intelligence* (AAAI), Philadelphia, PA, August 1986.

Koomen, J.A.G.M., "Reasoning about recurrence," Ph.D. Thesis and TR 307, Computer Science Dept., U. Rochester, July 1989; condensed version to appear, *Int'l. J. of Intelligent Systems* (Special Issue on Time in Artificial Intelligence), 1990.

Kyburg, H.E., Jr., "Bets and beliefs," reprinted in P. Gärdenfors and N.-E. Sahlin (Eds). *Decision, Probability, and Utility: Selected Readings.* Cambridge: Cambridge U. Press, 1988.

Kyburg, H.E., Jr., "Ellery Eels, rational decision and causality, and Paul Horwich, probability and evidence," *Int'l. Studies in Philosophy 19*, 1, 72-74, 1987.

Kyburg, H.E., Jr., "Epistemologial relevance and statistical knowledge," TR 251, Computer Science Dept., U. Rochester, April 1988.

Kyburg, H.E., Jr., "Full belief," TR 245, Computer Science Dept., U. Rochester, February 1988.

Kyburg, H.E., Jr., "Getting fancy with uncertainty," invited paper, *Society for Exact Philosophy*, Newfoundland, September 1987.

Kyburg, H.E., Jr., "Higher order probabilities," *Proc., 1987 AAAI Workshop on Uncertainty in Artificial Intelligence*, 30-38, July 1987.

Kyburg, H.E., Jr., "Higher order probabilities and intervals," invited talk, *1987 AAAI Workshop on Uncertainty in Artificial Intelligence*, July 1987; TR 236, Computer Science Dept., U. Rochester, November 1987.

Kyburg, H.E., Jr., "Knowledge," in J.F. Lemmer and L.N. Kanal (Eds). *Uncertainty in Artificial Intelligence*, Vol. 2 (1986 Workshop). Amsterdam: North-Holland, 263-272, 1988.

Kyburg, H.E., Jr., "Objective probabilities," *Proc., Int'l. Joint Conf. on Artificial Intelligence*, 902-904, August 1987.

Kyburg, H.E., Jr., "Objective probability," *Proc., 1987 AAAI Workshop on Uncertainty in Artificial Intelligence*, 148-155, July 1987.

Kyburg, H.E., Jr., "Probabilistic inference," TR 230, Computer Science Dept., U. Rochester, November 1987.

Kyburg, H.E., Jr., "Probabilistic inference and non-monotonic inference," TR 249, Computer Science Dept., U. Rochester, April 1988.

Kyburg, H.E., Jr., "Probabilistic inference and probabilistic reasoning," TR 248, Computer Science Dept., U. Rochester, April 1988.

Kyburg, H.E., Jr., "Representing knowledge and evidence for decision," in B. Bouchon and R.R. Yager (Eds). *Uncertainty in Knowledge-Based Systems* (Lecture Notes in Computer Science). Berlin: Springer Verlag, 30-40, 1987.

LeBlanc, T.J., "Parallel program debugging," *Proc., 13th Annual IEEE Int'l. Computer Software and Applications Conf.* (COMPSAC 89), Orlando, FL, September 1989.

LeBlanc, T.J., "Problem decomposition and communication tradeoffs in a shared-memory multiprocessor," in M. Schultz (Ed). *Numerical Algorithms for Modern Parallel Computer Architectures* (IMA Volumes in Mathematics and its Applications, Vol. 13). Springer-Verlag, 145-163, 1988a.

LeBlanc, T.J., "Shared memory versus message-passing in a tightly-coupled multiprocessor: A case study," *Proc., Int'l. Conf. on Parallel Processing*, 463-466, August 1986; BPR 3 (revised), Computer Science Dept., U. Rochester, February 1987.

LeBlanc, T.J., "Structured message passing on a shared-memory multiprocessor," *Proc., 21st Annual Hawaii Int'l. Conf. on System Sciences*, 188-194, January 1988b.

LeBlanc, T.J., "Structured message passing on a shared-memory multiprocessor," in P.J. Waterman (Ed.). *Applications on the Butterfly Parallel Processor. Research Monographs on Parallel and Distributed Computing.* London: Pitman Publishing/MIT Press, to appear, 1990.

LeBlanc, T.J. and S.A. Friedberg, "Hierarchical process composition in distributed operating systems," *Proc., 5th Int'l. Conf. on Distributed Computing Systems*, 26-34, May 1985a.

LeBlanc, T.J. and S.A. Friedberg, "HPC: A model of structure and change in distributed systems," *IEEE Trans. on Computers C-34*, 12, 1114-1129, December 1985; TR 153, Computer Science Dept., U. Rochester, May 1985b.

LeBlanc, T.J., N.M. Gafter, and T. Ohkami, "SMP: A message-based programming environment for the BBN Butterfly," BPR 8, Computer Science Dept., U. Rochester, July 1986.

LeBlanc, T.J. and S. Jain, "Crowd control: Coordinating processes in parallel," *Proc., Int'l. Conf. on Parallel Processing*, 81-84, August 1987.

LeBlanc, T.J., B.D. Marsh, and M.L. Scott, "Memory management for large-scale NUMA multiprocessors," TR 311, Computer Science Dept., U. Rochester, March 1989a.

LeBlanc, T.J. and J.M. Mellor-Crummey, "Debugging parallel programs with instant replay," *IEEE Trans. on Computers C-36* (Special Issue on Parallel and Distributed Computation), 4, 471-482, April 1987; TR 194 and BPR 12, Computer Science Dept., U. Rochester, September 1986.

LeBlanc, T.J. and J.M. Mellor-Crummey, "Debugging parallel programs with instant replay," in P.J. Waterman (Ed.). *Applications on the Butterfly Parallel Processor. Research Monographs on Parallel and Distributed Computing.* London: Pitman Publishing/MIT Press, to appear, 1990.

LeBlanc, T.J., J.M. Mellor-Crummey, and R.J. Fowler, "Using multiple views for parallel program analysis," to appear, *J. of Parallel and Distributed Computing*, June 1990.

LeBlanc, T.J., J.M. Mellor-Crummey, N.M. Gafter, L.A. Crowl, and P.C. Dibble, "The Elmwood multiprocessor operating system," *Software—Practice and Experience 19*, 11, 1029-1056, November 1989b.

LeBlanc, T.J. and B.P. Miller (Eds.), Summary of the *ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, Operating Systems Review* 22, 4, 7-19, October 1988; *SIGPLAN Notices 24*, 1, ix-xxi, January 1989.

LeBlanc, T.J., M.L. Scott, and C.M. Brown, "Large-scale parallel programming: Experience with the BBN Butterfly," *Proc., ACM SIGPLAN Symp. on Parallel Programming: Experience with Applications, Languages, and Systems*, New Haven, CT (July 1988), *SIGPLAN Notices 23*, 9, 161-172, September 1988; BPR 22, Computer Science Dept., U. Rochester, September 1988.

Litman, D.J., "Plan recognition and discourse analysis: An integrated approach for understanding dialogues," Ph.D. Thesis and TR 170, Computer Science Dept., U. Rochester, 1985.

Low, J.R., "Experiments with remote procedure call on the Butterfly," BPR 16, Computer Science Dept., U. Rochester, December 1986.

Low, J.R., "Lexical analysis on a moderately sized multiprocessor," TR 261, Computer Science Dept., U. Rochester, October 1988.

Lynne, K.J., "Competitive reinforcement learning," *Proc., Fifth Int'l. Conf. on Machine Learning*, 188-199, June 1988.

Mailloux, G.E., F. Langlois, P.Y. Simard, and M. Bertrand, "Analysis of heart motion from two-dimensional echocardiograms by velocity field decomposition," *Computer in Cardiology*, Leuven, IEEE Computer Society Press, October 1987.

Mailloux, G.E., F. Langlois, P.Y. Simard, and M. Bertrand, "Restoration of the velocity field of the heart from two-dimensional echocardiograms," *IEEE Trans. Medical Imaging 8*, 2, 143-153, 1989.

Marsh, B.D., "Psyche: a NUMA operating system kernel," in TR 88-03 (Proc., 3rd Annual U. Buffalo Graduate Conf. on Computer Science), Computer Science Dept., SUNY Buffalo, March 1988.

Martin, N.G., "Abstraction in planning: A probabilistic approach," Internal Report, Computer Science Dept., U. Rochester, December 1989.

Martin, N.G., J.F. Allen, and C.M. Brown, "ARMTRAK: A domain for the unified study of natural language, planning, and active vision," TR 324, Computer Science Dept., U. Rochester, January 1990.

Mellor-Crummey, J.M., "Concurrent queues: Practical fetch-and-ϕ algorithms," TR 229, Computer Science Dept., U. Rochester, October 1987.

Mellor-Crummey, J.M., "Debugging and analysis of large-scale parallel programs," Ph.D. Thesis and TR 312, Computer Science Dept., U. Rochester, September 1989a.

Mellor-Crummey, J.M., "Designing concurrent data structures for correctness," Working Paper, Computer Science Dept., U. Rochester, June 1988a.

Mellor-Crummey, J.M., "Experiences with the BBN Butterfly," invited paper, *Proc., COMPCON Spring '88*, 101-104, February 1988b.

Mellor-Crummey, J.M., "The 1988 International Conference on Fifth Generation Computer Systems," trip report, *Scientific Information Bulletin 14*, 4, October - December 1989b.

Mellor-Crummey, J.M. and T.J. LeBlanc, "A software instruction counter," *Proc., 3rd Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems*, 78-86, Boston, MA, April 1989.

Mellor-Crummey, J.M., T.J. LeBlanc, L.A. Crowl, N.M. Gafter, and P.C. Dibble, "Elmwood—An object-oriented multiprocessor operating system," TR 226 and BPR 20, Computer Science Dept., U. Rochester, September 1987; *1987-88 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, October 1987.

Miller, B.W., "The RHET Plan Recognition System, Version 1.0," TR 298, Computer Science Dept., U. Rochester, January 1990.

Mulac, J. and D. Baldwin, "Local propagation as a constraint satisfaction technique," TR 265, Computer Science Dept., U. Rochester, January 1989.

Nelson, R.C., "Using flow field divergence for obstacle avoidance: Towards qualitative vision," *Proc., Int'l. Conf. on Computer Vision*, 188-196, Tampa, FL, December 1988a.

Nelson, R.C., "Visual homing using an associative memory," *AAAI 1989 Spring Symp. Series*, Stanford, CA, March 1989; *Proc., DARPA Image Understanding Workshop*, Palo Alto, CA, May 1989.

Nelson, R.C., "Visual navigation," Ph.D. Thesis and TR, Computer Science Dept., U. Maryland, August 1988b.

Nelson, R.C. and J. Aloimonos, "Determining motion parameters from spherical flow fields (or the advantages of having eyes in the back of your head)," *Biological Cybernetics 58*, 261-273, 1988.

Nelson, R.C., and J. Aloimonos, "Obstacle avoidance using flow field divergence," *IEEE Trans. PAMI 11*, 10, 1102-1106, October 1989.

Nelson, R.C., D.H. Ballard, and S.D. Whitehead, "Visual behavior and intelligent agents," *Proc., SPIE Symp. on Advances in Intelligent Robotics Systems*, Philadelphia, PA, November 1989.

Olson, T.J., "An architectural model of visual motion understanding," Ph.D. Thesis and TR 305, Computer Science Dept., U. Rochester, August 1989.

Olson, T.J., "Finding lines with the Hough transform on the BBN Butterfly parallel processor," BPR 10, Computer Science Dept., U. Rochester, September 1986c.

Olson, T.J., "An image processing package for the BBN Butterfly parallel processor," BPR 9, Computer Science Dept., U. Rochester, September 1986b.

Olson, T.J., "Modula-2 on the BBN Butterfly multiprocessor," BPR 4, Computer Science Dept., U. Rochester, January 1986a.

Olson, T.J., L. Bukys, and C.M. Brown, "Low-level image analysis on an MIMD architecture," *Proc., First IEEE Int'l. Conf. on Computer Vision*, 468-475, London, England, June 1987.

Olson, T.J. and R.D. Potter, "Real-time vergence control," TR 264, Computer Science Dept., U. Rochester, November 1988; *Proc., Computer Society Conference on Computer Vision and Pattern Recognition*, San Diego, CA, June 1989.

Pelavin, R.N., "A formal approach to planning with concurrent actions and external events," Ph.D. Thesis and TR 254, Computer Science Dept., U. Rochester, May 1988.

Porat, S., "Fairness in context-free grammars under every choice of strategy," to appear, *Information and Control*, 1988b.

Porat, S., "Infinite behavior in connectionist models with asymmetric weights," *1987-88 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, October 1987; revised version in R. Trappl (Ed). *Cybernetics and Systems '88* (Proc., 9th European Meeting on Cybernetics and Systems Research). Dordrecht: Kluwer Academic Publishers, 1023-1030, 1988a.

Porat, S. and J.A. Feldman, "Learning automata from ordered examples," TR 241, Computer Science Dept., U. Rochester, April 1988.

Puckett, D.L. and P.C. Dibble. *The Complete Rainbow Guide to OS-9 Level II, Vol. 1: A Beginner's Guide to Windows*. Prospect, KY: Falsoft Inc., 1987.

Quiroz. C.A. (Ed.), "Proceedings of 1988 Open House," TR 209, Computer Science Dept., U. Rochester, May 1988.

Rashid, R.J. and G.G. Robertson, "Accent: A communication oriented network operating system kernel," *Operating Systems Review 15*, 5, 1981.

Rimey, R.D. and C.M. Brown, "Selective attention as sequential behavior: Modelling eye movements with an augmented hidden Markov model," TR 327, Computer Science Dept., U. Rochester, February 1990.

Scott, M.L. "The interface between distributed operating system and high-level programming language," *Proc., Int'l. Conf. on Parallel Processing*, 242-249, St. Charles, IL, August 1986; also appeared as BPR 6, TR 182, and in *1986-87 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, September 1986a.

Scott, M.L., "Language support for loosely-coupled distributed programs," *IEEE Trans. on Software Engineering SE-13* (Special Issue on Distributed Systems), 1, 88-103, January 1987.

Scott, M.L., "LYNX Reference Manual," (revised) BPR 7, Computer Science Dept., U. Rochester, August 1986b.

Scott, M.L., "An overview of Lynx," TR 308, Computer Science Dept., U. Rochester, August 1989; submitted for publication.

Scott, M.L. and A.L. Cox, "An empirical study of message-passing overhead," BPR 17, Computer Science Dept., U. Rochester, December 1986; *Proc., 7th Int'l. Conf. on Distributed Computing Systems*, 536-543, September 1987.

Scott, M.L. and R.A. Finkel, "A simple mechanism for type security across compilation units," *IEEE Trans. on Software Engineering 14*, 8, 1238-1239, August 1988.

Scott, M.L. and K.R. Jones, "Ant Farm: A lightweight process programming environment," BPR 21, Computer Science Dept., U. Rochester, August 1988; invited chapter in P.J. Waterman (Ed.). *Applications on the Butterfly® Parallel Processor.* London: Pitman Publishing / MIT Press, April 1989.

Scott, M.L. and T.J. LeBlanc, "Psyche: A general-purpose operating system for shared-memory multiprocessors," BPR 19, Computer Science Dept., U. Rochester, July 1987; TR 223, Computer Science Dept., U. Rochester, August 1987.

Scott, M.L., T.J. LeBlanc, and B.D. Marsh, "Design rationale for Psyche, a general-purpose multiprocessor operating system," *Proc., 1988 Int'l. Conf. on Parallel Processing*, 255-262, Vol. II (Software), St. Charles, IL, August 1988; *1988-89 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, October 1988.

Scott, M.L., T.J. LeBlanc, and B.D. Marsh, "Evolution of an operating system for large-scale shared-memory multiprocessors," TR 309, Computer Science Dept., U. Rochester, March 1989a.

Scott, M.L., T.J. LeBlanc, and B.D. Marsh, "Implementation issues for the Psyche multiprocessor operating system," *Proc., First Workshop on Experiences with Building Distributed and Multiprocessor Systems*, 227-236, Ft. Lauderdale, FL, October 1989c.

Scott, M.L., T.J. LeBlanc, and B.D. Marsh, "Multi-model parallel programming in Psyche," *Proc., 2nd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming* (PPoPP), Seattle, WA, to appear, March 1990a.

Scott, M.L., T.J. LeBlanc, and B.D. Marsh, "A multi-user, multi-language open operating system," *Proc., 2nd Workshop on Workstation Operating Systems*, 125-129, Pacific Grove, CA, September 1989b.

Scott, M.L., T.J. LeBlanc, B.D. Marsh, T.G. Becker, C. Dubnicki, E.P. Markatos, and N.G. Smithline, "Implementation issues for the Psyche multiprocessor operating system," submitted for journal publication, 1990b.

Scott, M.L. and S.-K. Yap, "A grammar-based approach to the automatic generation of user-interface dialogues," *Proc., 1988 ACM Conf. on Human Factors in Computing Systems* (CHI '88), 73-78, May 1988.

Shastri, L., "Evidential reasoning in semantic networks: a formal theory and its parallel implementation," Ph.D. thesis and TR 166, Computer Science Dept., U. Rochester, September 1985.

Shastri, L. and J.A. Feldman, "Neural nets, routines and semantic networks," in N. Sharkey (ed). *Advances in Cognitive Science.* Ellis Horwood Publishers, 1986.

Sher, D.B., "Evidence combination for vision, using likelihood generators," *Proc., DARPA Image Understanding Workshop*, December 1985a.

Sher, D., "Generating robust operators from specialized ones," *Proc., IEEE Workshop on Computer Vision*, November 1987a.

Sher, D.B., "Optimal likelihood generators for edge detection under Gaussian additive noise," *Proc., IEEE Conf. on Computer Vision and Pattern Recognition*, June 1986.

Sher, D.B., "A probabilistic analysis of template matching," Working Paper, Computer Science Dept., U. Rochester, Summer 1987b.

Sher, D., "A probabilistic approach to low-level vision," Ph.D. Thesis and TR 232, Computer Science Dept., U. Rochester, October 1987c.

Sher, D.B., "Template matching on parallel architectures," TR 156, Computer Science Dept., U. Rochester, July 1985b.

Sher, D., "Tunable facet model likelihood generators for boundary pixel detection," *Proc., IEEE Workshop on Computer Vision*, November 1987d.

Sher, D.B. and A. Tevanian, "The vote tallying chip: A custom integrated circuit," TR 144, Computer Science Dept., U. Rochester, November 1984.

Simard, P.Y., "Using recurrent networks to learn sequences in time," invited critical review, to appear, *Neural Network Review*, 1990.

Simard, P.Y. and G.E. Mailloux, "A projection operator for the restoration of divergence-free vector fields," *IEEE Trans. PAMI 10*, 2, 248-256, March 1988.

Simard, P.Y. and G.E. Mailloux, "Vector field restoration by the method of convex projections," to appear, *Computer Vision, Graphics, and Image Processing*, 1990.

Simard, P.Y., M. Ottaway, and D.H. Ballard, "Analysis of recurrent backpropagation," *Snowbird Conf. on Neural Networks*, April 1988; TR 253, Computer Science Dept., U. Rochester, July 1988; *Carnegie Mellon University Connectionist Models Summer School* (June 1988), December 1988;.also appeared as "Fixed point analysis for recurrent networks," *Proc., IEEE Conf. on Neural Information Processing Systems* (NIPS), 149-159, Denver, CO, December 1988.

Small, S.L., L. Shastri, M.L. Brucks, S.G. Kaufman, G.W. Cottrell, & S. Addanki, "ISCON: A network construction aid and simulator for connectionist models," TR 109, Computer Science Dept., U. Rochester, April 1983.

Swain, M.J., "Comments on Samal and Henderson: 'Parallel consistent labeling algorithms'," *Int'l. J. Parallel Programming 17*, 6, December 1988a.

Swain, M.J., "Object recognition from a large database," in TR 209 (Proc., 1988 Open House), Computer Science Dept., U. Rochester, May 1988b.

Swain, M.J., "Object recognition from a large database using a decision tree," *Proc., DARPA Image Understanding Workshop*, April 1988c.

Swain, M.J. and P.R. Cooper, "Parallel hardware for constraint satisfaction," *Proc., DARPA Image Understanding Workshop*, 620-624, April 1988; *Proc., Nat'l. Conf. on Artificial Intelligence* (AAAI), St. Paul, MN, August 1988.

Swain, M.J. and L.E. Wixson, "Efficient estimation for Markov random fields," *Proc., Image Understanding and Machine Vision, Topical Meeting of the Optical Society of America*, N. Falmouth, MA, June 1989.

Swain, M.J., L.E. Wixson, and P.B. Chou, "Efficient parallel estimation for Markov random fields," *Proc., 5th Workshop on Uncertainty and Artificial Intelligence*, Windsor, Ontario, August 1989.

Tenenberg, J.D., "Abstraction in planning," Ph.D. Thesis and TR 250, Computer Science Dept., U. Rochester, May 1988.

Tilley, D.G., "Zebra for MaxVideo: An application of object oriented microprogramming to register level devices," TR 315, Computer Science Dept., U. Rochester, November 1989.

Waltz, D. and J.A. Feldman (Eds). *Connectionist Models and their Applications*. Norwood, NJ: Ablex Publishing Company, 1988.

Watts, N., "Calculating the principal views of a polyhedron," TR 234, Computer Science Dept., U. Rochester, December 1987; *Proc., 9th Int'l. Conf. on Pattern Recognition*, Vol. 1, 316-322, Rome, November 1988.

Weber, J.C., "A parallel algorithm for statistical belief refinement and its use in causal reasoning," *Proc., Int'l. Joint Conf. on Artificial Intelligence*, Detroit, MI, August 1989.

Weber, J.C., "Principles and algorithms for causal reasoning with uncertainty," Ph.D. Thesis and TR 287, Computer Science Dept., U. Rochester, May 1989.

Weber, S. Hollbach, "A structured connectionist approach to direct inferences and figurative adjective-noun combinations," Ph.D. Thesis and TR 289, Computer Science Dept., U. Rochester, May 1989.

Whitehead, S.D., "Thesis proposal: Scaling reinforcement learning systems," Thesis Proposal and TR 304, Computer Science Dept., U. Rochester, August 1989.

Whitehead, S.D. and D.H. Ballard, "Active perception and reinforcement learning," TR 331, Computer Science Dept., U. Rochester, February 1990; submitted for conference publication.

Whitehead, S.D. and D.H. Ballard, "Connectionist designs on planning," *Proc., Carnegie Mellon University Connectionist Models Summer School*, June 1988.

Wixson, L.E., "Acquiring, representing, and using high-level knowledge about spatial relationships to guide a mobile camera," forthcoming TR, Computer Science Department, U. Rochester, to appear, 1990.

Wixson, L.E. and D.H. Ballard, "Real-time detection of multi-colored objects," in P.S. Schenker (Ed.). *Sensor Fusion II: Human and Machine Stategies (SPIE Vol. 1198)* (Philadelphia, PA, November 1989), 435-446, 1990.

Yamauchi, B., "JUGGLER: Real-time sensorimotor control using independent agents," *Proc., Optical Society of America Image Understanding and Machine Vision Conf.*, N. Falmouth, MA, June 1989.

Yap, S.-K., "DVI previewers," *Proc., 9th TeX Users Group Meeting*, 211-218, Montreal, August 1988a.

Yap, S.-K., "PENGUIN: A language for specifying user interface dialogues," in TR 209 (Proc., 1988 Open House), Computer Science Dept., U. Rochester, May 1988b.

Yap, S.-K. and A. Albicki, "Covering a set of test patterns by a cellular automaton," *1987-88 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, October 1987.

Yap, S.-K. and J. Kalinowski, "Experiences with the X Window System for computer aided self-testing designs," *Proc., 30th Midwest Symposium for Circuits and Systems*, August 1987.

Yap, S.-K. and M.L. Scott, "PenGuin: A language for reactive graphical user interface programming," to appear, INTERACT '90, Cambridge, United Kingdom, 1990.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.