NASA/CR-96- 206090

IN-74-CR
OCIT
[illegible]

A Scalable Distributed Approach to Mobile Robot Vision (NAG 9-828) 9-1-1995 to 8-31-1996 Final Report

Benjamin Kuipers, Principal Investigator
Robert L. Browning, Graduate Research Assistant
William S. Gribble, Graduate Research Assistant

April 30, 1997

Abstract

This paper documents our progress during the first year of work on our original proposal entitled "A Scalable Distributed Approach to Mobile Robot Vision".

We are pursuing a strategy for real-time visual identification and tracking of complex objects which does not rely on specialized image-processing hardware. In this system *perceptual schemas* represent objects as a graph of *primitive features*. Distributed software agents identify and track these features, using variable-geometry image subwindows of limited size. Active control of imaging parameters and selective processing makes simultaneous real-time tracking of many primitive features tractable. Perceptual schemas operate independently from the tracking of primitive features, so that real-time tracking of a set of image features is not hurt by latency in recognition of the object that those features make up. The architecture allows semantically significant features to be tracked with limited expenditure of computational resources, and allows the visual computation to be distributed across a network of processors. Early experiments are described which demonstrate the usefulness of this formulation, followed by a brief overview of our more recent progress (after the first year).

Keywords: Active vision, vision architectures, object recognition.

1 Introduction

1.1 Active Perception

The active approach to computer vision exploits the observation that certain visual processing tasks are fundamentally easier when performed within a control loop involving both the environment and the perceiver. Active vision employs a variety of techniques to effectively manage the complexity of the visual signal without complex representations of the physical world or complete, high resolution sampling of the entire visual field.

An active vision system may limit its visual processing to regions of the visual field that are expected to have information relevant to the agent's current goals. This can be accomplished by varying the resolution at which the visual space is sampled or by limiting the spatial extent of the regions of interest. Both techniques can dramatically decrease the density of the visual input and hence the computational resources needed to process it. Active vision systems attempt to acquire only the information necessary for a given set of tasks and to process that information in a goal dependent fashion.

Decreasing the quantity of information that must be processed by these selective methods and tailoring the computations performed to reflect the nature of the given task increases the rate at which the system can process a given visual signal. This allows the system to respond more quickly to changes in the world; in the context of an embodied agent interacting with an unpredictable environment, long latencies in response to sensory inputs can lead to unacceptable performance.

1.2 Visual Tasks For Mobile Robots

Our approach to active vision is oriented toward solving problems related to mobile robot navigation, exploration, and map-building. The following tasks are typical of those required of a mobile robot vision system:

Landmark tracking. A mobile robot should be able to identify visual landmarks that are significant for navigation, such as strong vertical edges along a corridor, or the extreme point of a nearby convex corner. The tracking of landmarks fixed in world space allows the robot to infer its own motion and provides useful feedback to motion control laws [10].

Obstacle detection. A mobile robot must be able to detect objects that might impede its progress. Also, active (possibly hostile) agents such as moving graduate students and floor polishers should be detected and avoided.

Location of traversable spaces. As a complement to obstacle detection, the robot must be able to find spaces that it can navigate without obstruction.

Place recognition. The Spatial Semantic Hierarchy (SSH) framework for robot spatial reasoning [8] characterizes places by a measure of "distinctiveness." A robot vision system should notice when its current neighborhood is a distinctive space such as an intersection of hallways, a concave or convex wall corner, etc. It has been shown that characterization of such places allows the robot to build useful maps of its environment [13].

Identification of regularities. By exploring the relation between its actions and its sensory input, the robot should be able to identify domain-specific regularities that provide useful perceptual features [13]. For example, vertical edges are important features in many environments, while indoor office environments also include many horizontal edges.

In a complex environment there will generally be a number of visual tasks to be performed concurrently: landmarks to be tracked, places to be recognized, and a variety of obstacles of which the system must remain aware. This suggests a natural decomposition of the problem into a number of independent visual routines each of whose goal is the fulfillment of a given visual task.

In the next section we present such a decomposition. We model the types of visual tasks described above as problems in constraint satisfaction given a symbolic description of the primitive features of the scene. Generating such a symbolic description is a task-dependent process which is driven by the operation of simple reactive image feature trackers. Following the description of our philosophy, we discuss some of the relevant features of our implementation. We then discuss experimental results which suggest that our method may be applied to a range of real tasks in robot vision, and finish with a brief overview of our more recent progress.

During the course of this research project, the following papers were published:

- "ARGUS: A Distributed Environment for Real-Time Vision" [5]
- "Slow visual search in a fast-changing world" [6]
- "Dynamic binding of visual percepts for robot control" [9]

2 Concepts

2.1 A Hierarchy of Visual Routines

We propose a two-level hierarchy of representation which allows complex visual phenomena to be abstracted into symbols that the robot can manipulate more readily than raw image data. At the top level of the hierarchy, *perceptual schemas* perform model-based processing to detect and identify complex phenomena in the world. Underlying the perceptual schemas are *primitive trackers* which translate features of the raw data (*primitive features*) into a representation which can be used for the perceptual schema's model processing. In the following two sections, we briefly describe each of these levels and then discuss some of the implications for scalable distributed processing.

2.1.1 Perceptual Schemas

A perceptual schema can be regarded as a virtual sensor tuned to respond to a particular landmark, spatial property, or event of interest [1]. A robot may have many different perceptual schemas, any number of which could be active at a given time, depending on the robot's state and current goals. For instance, a robot might have a particular perceptual schema which recognizes its recharging station, another which recognizes open pathways in nearby space, and another which recognizes doors. The open-pathways schema would likely be always active, the door-detection schema active only when the robot is trying to find a door, and the charger-detecting schema active when the robot's batteries are running low.

Perceptual schemas have a current state expressed as a vector of characteristic values. These values include measures of the position, pose, and extent of the tracked feature, and an evaluation of the schema's confidence that it has accurately matched its model to an image-space artifact. This vector of characteristic values can be used by higher-level processing to reason about its environment.

An initial hypothesis of model detection can be made based on a match between a single primitive feature of the input image stream and a feature of the model. Confidence in a proper match increases as more primitive features and their spatial relationships are matched with the model. The process of incrementally matching the model against features of the image stream can be thought of as a model-directed exploration of the image stream [7].

For example, consider a perceptual schema tuned to detect doors in an office-like environment. A model of a door could contain primitive features including two long parallel sides connected by a straight (but not necessarily perpendicular, depending on viewing angle) edge at the top, height of about seven feet, height-to-width ratio of about 2:1, and a difference in depth between the edge features and the space between (assuming that only open doors are of interest). Such a model could be represented by a simple relational graph, or by a map of distinctive points and interconnections following the Spatial Semantic Hierarchy [8]. An initial hypothesis of "doorness" for a region of image space can be made upon successful location of a single model feature, such as a vertical line. The model of a door predicts that there will be an intersecting line at the top of a vertical line. If such a feature is found, the probability that a door has been found increases incrementally and an attempt is made to locate the next primitive feature of the model.

2.1.2 Primitive Feature Trackers And Visual Control Laws

Perceptual schemas utilize *primitive trackers* and their associated *visual control laws* to construct a representation for comparison with a model. Primitive trackers are the simplest level of interaction between the robot and the raw image stream. Each primitive tracker locates and tracks a single simple image feature over time.

A primitive tracker consists of a window on the image stream and a visual control law that reactively adjusts the parameters of the window. Primitive features are local features of image space, such as lines and corners, which occupy only a small portion of the robot's field of view. Therefore, they can be tracked without processing the entire image stream, by focusing attention on a small image window. In the case of a binocular or trinocular stereo system, the "image stream" could in fact be an aggregate stream of multiple image sources, with imaging parameters describing a multi-dimensional "window" that places an imaging window independently on each image stream to find corresponding features and determine vergence.

The visual control law determines what type of image feature the tracker is sensitive to and a policy for tracking that type of feature. A visual control law consists of a feature extraction step followed by a feature tracking step. Feature extraction is a composition of simple image processing operations generating candidate representations of the tracked image-space feature in the local window. Feature tracking is the determination of a best match among the candidate target features.

The window parameter adjustments that take place as a result of feature tracking do not necessarily leave the same portion of the feature under the tracking window. In certain situations, it is appropriate for the control law to "walk" the window along a spatially-extended feature such as an edge. This is a convenient way of dynamically exploring the relationships among spatially local features such as corners that define the extent of extended tracked features.

By separating visual processing into two weakly-coupled layers, real-time performance becomes primarily dependent on primitive feature tracking, with perceptual schema matching taking place in the background. Primitive feature tracking is tractable for three distinct reasons. First, each primitive feature is local in image space and processing can be restricted to small windows (currently about 1024 pixels or smaller). Second, features are tracked only when either triggered by a conspicuous event in the data ("pop-outs"), or when suggested by the model in a perceptual schema. And third, as discussed in the next section, the size and locality of primitive feature trackers allows them to be naturally allocated to simple parallel processors.

2.1.3 Implications For Scalable Processing.

Primitive trackers require only a small, localized segment of the raw image stream to perform their tracking operations and have no data dependencies on other trackers. Likewise, perceptual schemas depend only on a trickle of symbolic outputs from primitive trackers (a few words per frame describing the current tracker parameters). These properties have several important consequences for scalable processing. First, for a given number of visual routines, the computation can be performed by several small (and inexpensive) processors instead of one powerful processor. Second, the small data rate required means that trackers can be distributed over a network with modest bandwidth. Third, the separation of perceptual schemas from feature trackers means that robot performance degrades gracefully as computational resources are absorbed during activity.

Figure 1 shows a conceptual model of a robot vision system based on our architecture. Multiple perceptual schemas, each controlling multiple feature trackers, have a distributability that is relatively fine-grained with respect to the total number of tasks being performed. Any link between a perceptual

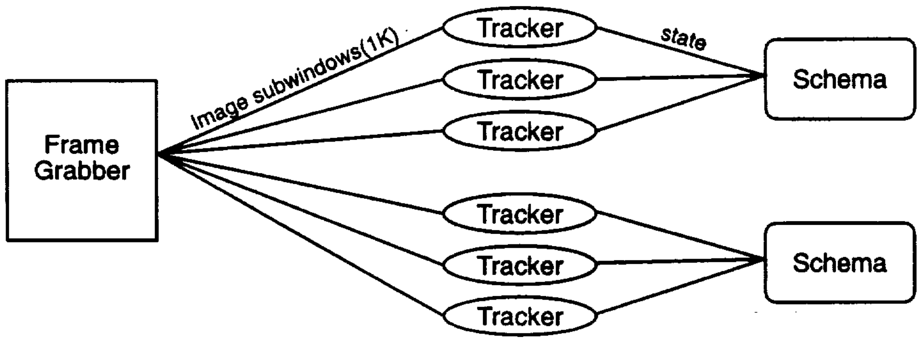

Figure 1: Data connections between schemas, trackers, and the image source.

schema and a feature tracker or between a feature tracker and its source of image data can be either a local-node or a network connection.

As described above, each primitive tracker requires an input of about 1 kbyte or less of data per frame (32 by 32 pixel window). For a real time vision system with a frame rate of 30 Hz, this is a total data bandwidth of 30 kbytes/sec per tracker. Our experiments have shown that simple features such as vertical lines can be reliably extracted and tracked with about 15 integer operations per pixel, for total compute resource requirement of .45 MIPS. This is well within the capabilities of cheaply available PC-class processors and DSP chips. A number of such chips supported by a network of several Mb/sec (comparable to workstation network interfaces) is capable of tracking primitive features numbering into the hundreds.

Distributed and parallel processing are not new ideas in the computer vision community [15, 12, 11, 3]. However, most efforts in multiprocessor vision have taken an approach which parallelizes a solution to a particular visual problem rather than exploiting the natural parallelism of visual tasks. While brute-force approaches to parallelization can be effective, a more elegant approach is to cast the problem in terms which let the parallelism fall out naturally. The semantics of perceptual schemas and primitive feature trackers allow visual routines to be expressed in a highly distributable form. Further, our experiments have indicated that a wide variety of real visual problems can be conveniently expressed in these terms.

A key feature of our approach is the graceful degradation of performance under load. Consider the door-recognizing schema described earlier. As it was described, such a schema would require several successive hypotheses to be confirmed before a reasonable match confidence was reached: finding an initial vertical line, then locating a corner, then another corner, and finally confirming a region of greater depth in between the two vertical lines. Due to its sequential nature, this process could take several image-frame times to complete, which amounts to a significant fraction of a second.

Humans perform quite well in tasks requiring recognition of these types of spatial relationships, and latencies of several hundred milliseconds are not unusual or particularly problematic [14]. A human can easily identify a door even if it is moving through the perceived image. Clearly, the process of recognizing the door is somewhat independent from the real-time task of tracking its position in the image. A computer vision system which depended on recognizing a door in each frame would not be able to function with such long latencies, since the door could have moved too far between image capture and recognition to make a dense-sampling assumption valid.

On the other hand, primitive feature tracking is a process that can be done in parallel with model matching, since each primitive tracker depends only on the raw image stream. While model-matching is taking place "in the background," the graph representation of the prospective landmark is dynamically maintained by the primitive trackers. Even if the latency between initial feature acquisition and final model match confirmation is many frame-times, the component primitive features of the prospective schema target will be tracked at the rate of incoming frames. Therefore, a hard limit on the time to complete a particular perceptual schema is determined by the demands of the robot's current task rather than by the necessity of model-matching for successful tracking. Adding more schemas for the robot to run concurrently will (up to the point where the system can no longer run its primitive trackers in real time) only cause a slowdown in the operation of perceptual schemas.

3 Implementation Details

3.1 Perceptual Schemas

ARGUS describes the visual structure of objects using *view templates*. A view template is a tree-structured description of the object's parts and their geometric relationships. Each part is described as an instance of a primitive image feature plus any details. Each detail is a subtree, which may have parts of its own. The geometric relations between object parts are encoded in *geometric constraints*. Geometric constraints have two parts: one or more *measures* of object properties and a *relation* between those measures. View templates are loaded and compiled dynamically by the system from expressions in a simple special-purpose language.

3.1.1 Measures

Measures are geometric operators which return a scalar, vector, or bounding-box value for a particular property of a single object part or of a relationship between several parts. Measures are a form of "instrumentation" placed on and between the parts of a model. Typical measures include `orientation`(x), `bounds-box`(x), `angle-between`(x, y), `size`(x), `center-point`(x), `aspect-ratio`(x), and `end-point`(x). In each case, the significant variables are subparts of the model.

3.1.2 Relations

Relations are real-valued functions which return a measure of agreement or disagreement (on the interval $[0, 1]$) with a predicate. Relations apply to one or more measures. Some relations understood by ARGUS include **equal-values**(x, y) (defined for scalar x and y, **coincident-points**(x, y) (defined for complex x and y), and **congruent-boxes**(x, y) (defined for bounding boxes x and y). The variables in relation statements are measures. In many cases, a relation is designed to compare a measure to a fixed value. Consider a relation that specifies two features f_1 and f_2 are perpendicular to each other. The constant measure $\frac{\pi}{2}$ is compared against the measured angle:

$$R = \texttt{equal-values}(\texttt{angle-between}(f_1, f_2), \frac{\pi}{2})$$

3.1.3 Projective invariance

View template models are a simple representation of objects as a constellation of visible features. These models do not directly represent the three-dimensional structure of objects, and so they are vulnerable to object self-occlusion and the other weaknesses of 2D representations of 3D objects. Ideally, geometric constraints are specified in such a way as to be invariant to affine distortions of the object's projected image, but the view-template framework does not enforce the restrictions on model construction and geometric constraint definition required to ensure affine invariance for the entire model. We have found that, in general, a straightforward representation of an object's geometry can be made invariant under image scaling and rotations both in and out of the imaging plane. When view template models are constructed for particular robot tasks, the applicability of specific models can be evaluated and multiple models of the same object can be used, if needed, to capture the range of possible viewpoints.

3.1.4 Visual search

Objects are located in the visual stream by a coarse-to-fine visual search augmented by feature-based return inhibition. Each type of primitive feature defines a mechanism for *salience map* generation. A salience map measures the likelihood that a particular image region is salient to the current search task. Salience maps are generated from a low-resolution attention buffer which spans the entire search region. The salience map search directs attention to objects which match the *key feature* representing the target object. For instance, a search for a green block might direct attention to any visible region of green pixels. Further model-matching is required to determine if the attended object matches the view template of the target. If it does not, that feature is added to an *inhibition list* of features known not to be the key feature of the actual object being searched for. The feature is still tracked, allowing a moving camera to successfully perform visual search without repeatedly returning to candidate objects that it has already rejected.

3.2 Primitive Feature Trackers

So far, we have implemented a small number of primitive feature trackers which, although designed for expedience rather than optimality, have served our purposes adequately. With only simple image-processing algorithms, the system can successfully track real, physical objects under conditions of unknown camera and object motion. With the addition of state-of-the-art image processing techniques, we expect that performance will be still better. ARGUS currently supports primitive feature trackers for either edges (lines) or colored blobs.

3.2.1 Edge Tracking

Vertical line tracking is a five-step process which the primitive feature tracker applies to each 1k-pixel frame. Horizontal line tracking is performed using the same computational functions but transposing the image before processing. Since only a tiny image buffer is processed and computationally inexpensive methods are used, vertical line tracking computations are very fast on typical computing hardware. Processing steps are:

Edge detection. The Sobel edge operator [4] is applied to get an estimate of the image intensity gradient. The Sobel operator was chosen for its computational simplicity and somewhat improved noise performance relative to simpler methods.

Vertical segment detection. A degenerate Hough transform [4] is applied to the gradient estimate image. The accumulator array is parameterized by horizontal and vertical position of short vertical line segments. Gradient direction and strength information are used to determine whether or not a particular pixel votes for a vertical line segment at each location. The number of accumulator bins that each pixel can vote for is limited to two, greatly decreasing the computational expense of this operation.

Segment linking. The accumulator array from the previous step is scanned for vertically adjacent strong segments. A list of extended segments with their positions and strengths is constructed.

Line matching. In its initial frame, the tracker finds the strongest vertical line segment in the image and records its position in the image. A PID controller predicts the next position of the tracked line segment. A simple distance metric finds the segment in the current frame most similar to the predicted line. If the tracker fails to find a sufficiently good match, it sets itself to a *lost* state and discontinues tracking.

Window adjustment. The tracker adjusts its window position, dimensions, and zoom in order to keep the predicted position of the tracked segment in the center of the window and to keep the entire vertical extent of the segment inside the window.

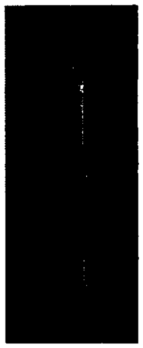

Figure 2: Edge tracker's view of the edge of a flash card.

Each tracker calculates a confidence measure which takes into account the length of time the line has been tracked, the strength of the line being tracked, and the length of line viewed. Maximum confidence is achieved when the tracked line is a line segment contained completely within the tracker's window. Window adjustments perform a simple hill-climbing operation to maximize the confidence by adjusting window size and zoom to get the best amount of the tracked line in the window. Figure 2 is a snapshot of an edge tracker's sub-window onto the image stream. The narrow inner outline outlines the edge within this sub-window that it is actually tracking.

Total computation involved in the line tracking operation is only a few tens of operations per image pixel. The line tracking routine runs at a rate of over 40 frames per second on a typical Sun SPARC workstation. The data rate of images to the tracker is set to 1 Kbyte or less per frame, depending on the window geometry.

3.2.2 Blob Tracker

The blob tracker is capable of tracking a single contiguous blob of similar color. Figure 3 shows the adaptive color tracker tracking a LEGO^{TM} block. The left image shows the attention buffer window used to track the block, and the right image shows a bitmap of the pixels that the tracking algorithm identifies with the object. Each incoming image is converted to a pixel-membership bitmap by a color oracle (described in Section 3.3), and the best-match contiguous blob (according to distance metrics similar to those used for the edge tracker) of acceptable color is tracked. Section 5 mentions some more recent improvements.

3.3 Color Oracles

To handle color, our feature trackers use a modular, color matching strategy, where color validity is determined by consulting an appropriate "color oracle".

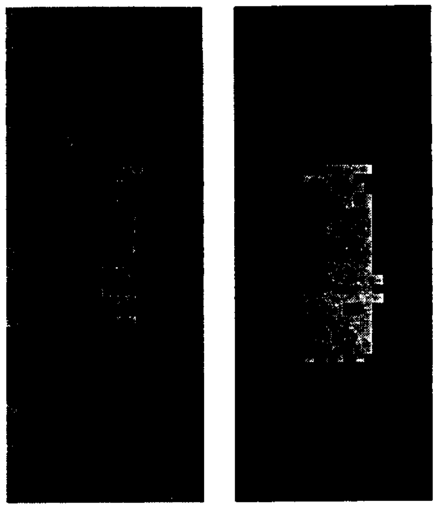

Figure 3: Adaptive-color tracker tracking a block. *left:* Attention buffer image. *right:* bitmap of pixels that are a part of the block.

A given tracker's oracle can weigh a number of factors to decide whether or not a given pixel belongs to the object. These factors might include the histogram of colors previously accepted, the geometric position of the pixel in question, or the color distance from other pixels in the object. At the end of the period covered by this report, oracles only used a fixed set of colors to make their determination of membership. Section 5 describes some more recent improvements.

3.4 Support for Scalable Processing

In ARGUS, connections between objects, like the connection between a feature tracker and its parent schema, may either be local connections with both components running on the same machine, or remote connections with each component running on a different machine. At the coding level, this distinction is hidden by a simple distributed object system which includes a Distributed Object Pre-Processor (DOPP), a service database, and a lower level support library.

Whenever an object providing a service is created, it registers itself with a service database. This database keeps track of what services are available and where. For example, when the object representing the frame grabber on a particular machine starts up, it registers itself as providing 'RGBMonocularVideo' on a particular port on its host machine. Later, when a new feature tracker needs an image stream, it consults the service database to determine where the 'RGBMonocularVideo' service is being provided, and to establish the connection to the server. This added level of indirection in locating the provider of a given service makes it easier to reconfigure the system (even while running), and makes it possible for the service database to perform some load-balancing functions.

To make coding our distributed objects easier, we developed DOPP, a preprocessor (currently for Scheme and C++ source) which automatically generates the support functions necessary to distribute a particular object class. Classes are defined normally, but annotated with special keywords indicating which methods should be available remotely. DOPP can then generate the underlying code necessary to create the proxy objects, instantiate the remote object servers, and handle object specific communications.

The most primitive foundation of our distributed object system is a small body of code providing the functionality which is independent of the particular class being distributed. This includes the service database, the distributed object server, the remote method server template, and the lowest level data communication functions.

3.5 RScheme

After developing the core vision technologies in C++, which was an appropriate language for that system's high computational demands, we decided that we needed a more flexible language for higher level control. At the higher levels, we felt power and expressiveness were significantly more important than speed. After some consideration we settled on Scheme, and began the process of augmenting a locally developed version, RSCHEME[1], for our specific needs.

In order to support our distributed object system, and in preparation for the more recent work on a control language described in Section 5, RSCHEME needed better support for threads, asynchronous network IO, and richer synchronization primitives. For several months we worked with the RSCHEME maintainers to add the required functionality to the language. In addition we translated the low-level primitives of our distributed object system to Scheme, and added support for Scheme code generation to DOPP.

We also created a Scheme library to control our Rhino arm. The Rhino is a Rhino Mark IV robot arm with a fixed base, five degrees of freedom, and a two finger gripper. The Scheme library handles distributed communication with the arm for issuing commands and collecting status updates.

4 Experiments

4.1 Tracking Flash Cards

Our first experimental apparatus consisted of an uncalibrated low-resolution monochrome CCD camera with a wide-angle lens, an Apple Macintosh 660 AV, and a Sun Sparc-10 workstation. Video frames were captured by the Macintosh, and appropriate subimages were extracted by a software agent and forwarded

[1]RSCHEME is a portable object-oriented Scheme system intended for use in language research projects implemented by Donovan Kolbly at the University of Texas at Austin, with direction from Prof. Paul Wilson. See <http://www.rosette.com/donovan/rs/>

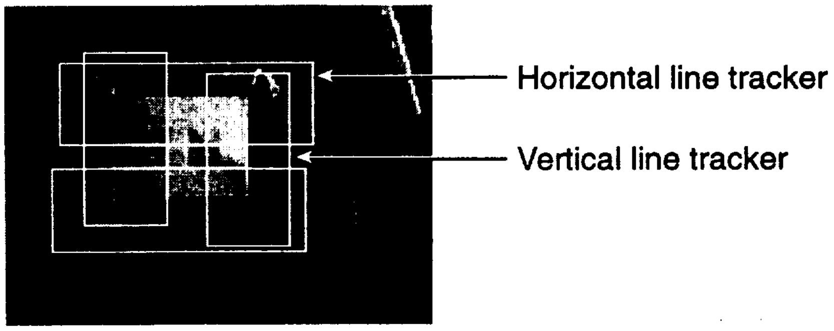

Figure 4: Perceptual schema tracking a real-world object, showing primitive feature windows.

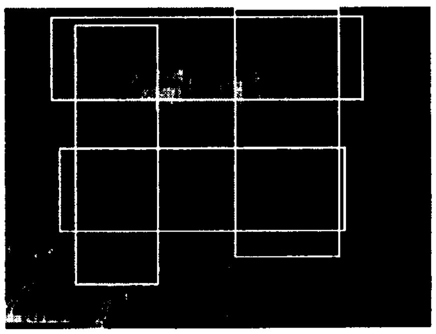

Figure 5: Perceptual schema tracking a less ideal rectangle.

across the departmental ethernet, which is shared by many other machines, to each active primitive feature tracker. Each extracted subimage was limited to 1k pixels in size, but the geometry and zoom factors were controllable by the tracker.

To test the effectiveness of the schema, we pointed the camera at an assortment of rectangular objects (see Figures 4 and 5). The relevant schema successfully identified rectangular shapes when presented with doors, windows, computer monitors, and sheets of paper. Typically the schema required twenty to fifty frames to completely construct its model and confirm that it was tracking a valid rectangle. It was tolerant of slight misalignment of straight lines and to the line-bowing distortions introduced by our camera. Camera movement and vibration during the recognition process did not adversely affect tracking, although it did slow down the recognition process as tracker uncertainties increased with rapid movements.

Due to problems with the Macintosh TCP/IP implementation, our frame rate was limited to 3 Hz. In tests using pre-captured images forwarded from one

Sun to another, real-time performance of 15 frames/second was achieved. Since total network traffic for the schema and four trackers is less than 10kBytes/frame, we expected that a very slight upgrade of our experimental apparatus would allow us to achieve real-time performance.

4.2 Tracking a Tennis Ball

Encouraged by our initial success with fairly limited hardware, we set out to migrate our system to new hardware, including a Dell Pentium running Linux containing a Matrox frame grabber, and a color camera. As we migrated the code, we also added a number of new features including the distributed object system and DOPP, support for color images, distributed access to the image stream, an interface to a 5-DOF RHINO XR-4 robot arm, and support for the schema model parser.

Once this work was completed, we set up an experiment where the robot arm (with the camera mounted on top in an eye-in-hand configuration) was intended to track a tennis ball pendulum swinging in front of it. We immediately discovered that our new system was quite an improvement over the old one. The TCP/IP limitations were gone, and we could easily track the tennis ball at over 10 fps, even when the ball was moving quite quickly in the camera's field of view. Unfortunately, limitations in the Rhino arm's control hardware, specifically latencies in command execution, prevented the arm from being able to follow the ball at reasonable speeds as it moved off the edges of the camera's view. Overall, though, the experiment was a success, proving that ARGUS could support visual tracking at real-time rates, and confirming that our newly added systems were functional.

4.3 Following a Blob

After discovering the limitations of the robot arm, we modified our experiment to accommodate those limitations while still closing the control loop. Our new goal was to track a blob of color moving around on the desktop while adjusting the position of the arm to keep the blob centered in the field of view.

The result was that the system was able to track a blob drawn on a piece of paper as it was moved fairly rapidly around around on the desktop. By this point we had acquired another Dell Pentium running Linux, and were also able to test our object distribution code. In further experiments the system was able to track several blobs simultaneously, with the trackers scattered across both machines, and with little noticeable degradation in performance.

5 Recent Progress and Future Plans

Under subsequent funding from NASA JSC[2] we have made significant improvements to our work which will be documented more fully in a future report, but are presented briefly here.

5.1 Dynamic Variables

For many robot tasks, the significant variables are derived from descriptions of physical objects which are within the robot's visual space. Interaction with a dynamic world requires that these descriptions be robust and updated in a timely fashion. For arm-like robot manipulators and for mobile robots, measures of the world's state can be thought of as *dynamic variables* which continuously change to reflect changes in the environment. Many sorts of time-dependent sensor inputs can be thought of as dynamic variables, including those for continuous motion control laws, for identification of objects and places, and for transition functions in a discrete-event model of the world. In this formulation, the task of robot sensory systems is to provide and maintain bindings for dynamic variables, which are symbolic representations of properties of the environment relevant to the robot's current action.

For example, a mobile robot trying to visually guide itself through a doorway needs to locate and track the edges of the door-frame; a robot arm grasping an object needs to know the position of its gripper and the object it is trying to grasp. The dynamic variables associated with these tasks are representations of the spatial and functional properties of the relevant objects as they relate to the robot's task.

To address this issue, we have added support for dynamic variables to our system. There is a straightforward method for describing these variables, and for indicating how to maintain the link between the symbolic representation of a visual percept and the dynamic image-space features it is founded upon. Though originally designed with visual applications in mind, this abstraction is also useful for other external sensors like sonar or even internal sensors like battery voltage or motor current.

5.2 The SPLAT Package

SPLAT [3] is designed to provide a structured framework for specifying robot actions. An individual splat defines a method or methods to achieve a given goal. The system depends on dynamic variables (and *slap fluents*) for its operation, and provides the control link between the continuously-updated measures of object properties and location generated by the dynamic variables, and the effectors that produce action in the world. The SPLAT framework is patterned

[2] "Spatial Reasoning for Scalable Distributed Mobile Robot Vision." (NAG 9-898.) Benjamin Kuipers, P.I. 8-6-96 to 7-31-97.

[3] Simple Provisional Language for Actions and Tasks

after that of the University of Chicago's RAPS [2], and leverages off of our improvements to RSCHEME to provide fully asynchronous, threaded behaviors.

5.3 Blob Tracker Improvements

Blob trackers now orient themselves to the major axis (if there is one) of the blob being tracked. This is accomplished through a simple moment calculation which keeps the vertical axis of the attention buffer aligned with the blob's longest axis, and provides some measure of the orientation of elongated objects.

5.4 Color Oracles

In our earlier system, oracles only used a fixed set of colors in making their determinations. We now have adaptive oracles which can discard inappropriate outlying pixels and add colors that are both similar to those already accepted by the oracle and which occur in regions where all the surrounding pixels are also already permitted. This has the effect of adapting to the color of a particular object and to smooth color variances over time. This has generally tightened the boundaries of the feature being tracked, and allows the system to handle more lighting variation.

5.5 Sorting Blocks

To test and demonstrate the functionality of our improved system, we designed a block sorting experiment. The goal is for the Rhino arm to sort a number of blocks into bins according to their color.

Figure 6 shows the robot and its workspace for these experiments. The camera is a low-cost CCD type with a fixed-focus lens on a mount attached to the wrist above the wrist-rotation joint. With only knowledge of the qualitative relationships between the sign of joint-motion direction and the image-space motion of an object at rest on the tabletop, the arm has been able to reliably sort the blocks into the appropriate bin.

5.6 Inexpensive Pan-Tilt-Vergence Platform

We have recently discovered an extremely flexible and inexpensive servo controller which should make it possible to construct a very inexpensive (probably less than $300) pan-tilt-vergence platform where each of the cameras is capable of independent or linked positioning. We should be constructing at least one of these in the near future.

5.7 A New Mobile Platform

In connection with our related research into assistive technologies for the disabled, we have purchased, and are awaiting delivery of a powered wheelchair with on-board sonar, IR sensors, and computer control. We plan to use this

Figure 6: Workspace for the LEGO^{TM} sorting task.

as the primary platform for our future vision research. Once the chair arrives, we will augment its current hardware with several PC-class motherboards, a two frame grabbers, and a pair of color cameras to make it a fully autonomous vision-based mobile platform.

6 Summary

We have described our distributed architecture for robot vision, ARGUS which exploits the natural parallelism of typical visual tasks. The main advantages of our approach are twofold. First, the tracking of primitive features is an operation that can often be performed at low-resolution, and requires only local image information and no interaction with other primitive feature trackers. This allows the trackers to be distributed across a network of workstations or inexpensive dedicated processors. Second, by separating higher-level model matching from primitive feature tracking, we permit real-time tracking in dynamic scenes without requiring a full cycle of schema completion for each frame.

ARGUS can track constellations of primitive image features which match tree-structured hierarchical models (perceptual schemas) of the objects represented by those features. These hierarchical models are constructed dynamically from descriptions which specify the topological structure of each model and the geometric constraints among the elements of that structure. ARGUS supports distributed computation through its distributed object system which, among other things, allows schemas and their constituent feature trackers to be transparently distributed across a collection of machines.

During development, we have performed a number of experiments to evaluate the system's emerging performance. Our earliest experiments on limited vision hardware convinced us that our approach was viable, and that the system should perform well on more reasonable hardware. Our later experiments validated that belief, demonstrating the system tracking objects in the field of view at real-time frame rates, and eventually closing the control loop by linking movement of a robot arm to the movement of tracked objects.

Finally, we have briefly discussed our more recent work which includes the creation a dynamic variable abstraction to facilitate linking symbolic control variables to dynamic visual phenomena, the development of a language tailored to the task of controlling robot actions, the completion of a new experiment which demonstrates good performance for a closed-loop sorting task, and the planned acquisition of some new hardware, including a powered wheelchair and an inexpensive pan-tilt-vergence platform.

References

[1] Ronald C. Arkin and Douglas MacKenzie. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Trans. on Robotics and Automation*, 10(3):276–286, 1994.

[2] R. Peter Bonasso, David Kortenkamp, David P. Miller, and Mark Slack. Experiences with an architecture for intelligent, reactive agents. In *Proc. 14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, 1995.

[3] Kevin J. Bradshaw, Phillip F. McLauchlan, Ian D. Reid, and David W Murray. Saccade and pursuit on an active head/eye platform. *Image and Vision Computing*, 12(3):155–163, 1994.

[4] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley, New York, 1992.

[5] William S. Gribble. Argus: a distributed environment for real-time vision. Master's thesis, University of Texas at Austin, Austin, Texas, 1995.

[6] William S. Gribble. Slow visual search in a fast-changing world. In *Proceedings of the 1995 IEEE Symposium on Computer Vision (ISCV-95)*, 1995.

[7] B. J. Kuipers. A frame for frames: representing knowledge for recognition. In D. G. Bobrow and A. Collins, editors, *Representation and Understanding*, pages 151–184. Academic Press, New York, 1975.

[8] B. J. Kuipers and Tod Levitt. Navigation and mapping in large scale space. *AI Magazine*, 9(2):25–43, 1988. Reprinted in Advances in Spatial Reasoning, Volume 2, Su-shing Chen (Ed.), Norwood NJ: Ablex Publishing, 1990.

[9] Benjamin Kuipers, William Gribble, and Robert Browning. Dynamic binding of visual percepts for robot control, 1997.

[10] Wan Yik Lee. Abstract mapping senses and a heterogeneous control based on landmark vector sense. unpublished paper, August 1992.

[11] M. Mirmehdi and T. J. Ellis. Parallel approach to tracking edge segments in dynamic scenes. *Image and Vision Computing*, 11(1):35–47, 1993.

[12] Nikolaos P. Papanikolopoulos and Pradeep K. Khosla. Adaptive robotic visual tracking: Theory and experiments. *IEEE Trans. on Automatic Control*, 38(3):429–445, March 1993.

[13] D. Pierce and B. Kuipers. Learning to explore and build maps. In *Proc. 12th National Conf. on Artificial Intelligence (AAAI-94)*. AAAI/MIT Press, 1994.

[14] Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984.

[15] P. H. Welch and D. C. Wood. Image tracking in real–time: a transputer emulation of some early mammalian vision processes. *Image and Vision Computing*, 11(4):221–228, 1993.