GARGOYLE: An Environment for Real-Time, Context-Sensitive Active Vision

Peter N. Prokopowicz, Michael J. Swain, R. James Firby, and Roger E. Kahn

Department of Computer Science
University of Chicago
1100 East 58th Street
Chicago, IL 60637
peterp@cs.uchicago.edu, swain@cs.uchicago.edu, firby@cs.uchicago.edu, kahn@cs.uchicago.edu

Abstract

Researchers in robot vision have access to several excellent image processing packages (e.g., Khoros, Vista, Susan, MIL, and XVision to name only a few) as a base for any new vision software needed in most navigation and recognition tasks. Our work in automonous robot control and human-robot interaction, however, has demanded a new level of run-time flexibility and performance: on-the-fly configuration of visual routines that exploit up-to-the-second context from the task, image, and environment. The result is Gargoyle: an extendible, on-board, real-time vision software package that allows a robot to configure, parameterize, and execute image-processing pipelines at run-time. Each operator in a pipeline works at a level of resolution and over regions of interest that are computed by upstream operators or set by the robot according to task constraints. Pipeline configurations and operator parameters can be stored as a library of visual methods appropriate for different sensing tasks and environmental conditions. Beyond this, a robot may reason about the current task and environmental constraints to construct novel visual routines that are too specialized to work under general conditions, but that are well-suited to the immediate environment and task. We use the RAP reactive plan-execution system to select and configure pre-compiled processing pipelines, and to modify them for specific constraints determined at run-time.

Introduction

The Animate Agent Project at the University of Chicago is an on-going effort to explore the mechanisms underlying intelligent, goal-directed behavior. Our research strategy is centered on the development of an autonomous robot that performs useful tasks in a real environment, with natural human instruction and feedback, as a way of researching the links between perception, action, and intelligent control. Robust and timely perception is fundamental to the intelligent behavior we are working to achieve.

As others have done before us (Bajcsy 1988; Ullman 1984; Chapman 1991; Ballard 1991; Aloimonos 1990), we have observed that a tight link between the perceptual and control systems enables perception to be well tuned to the context: the task, environment, and state of the perceiving agent (or robot in our case). As a result, perception can be more robust and efficient, and in addition these links can provide elegant solutions to issues such as grounding symbols in plans of the control system.

Our computer vision research concerns the problems of *identifying* relevant contextual constraints that can be brought to bear on our more or less traditional computer vision problems, and *applying* these constraints effectively in a real-time system. We have demonstrated that certain vision problems which have proven difficult or intractable can be solved robustly and efficiently if enough is known about the specific contexts in which they occur (Prokopowicz, Swain, & Kahn 1994; Firby *et al.* 1995). This context includes what the robot is trying to do, its current state, and its knowledge of what it expects to see in this situation. The difficulty lies not so much identifying the types of knowledge that can be used in different situations, but in applying that knowledge to the quick and accurate interpretation of images.

Our most important techniques in this regard so far have been the use of several low-level image cues to select regions of interest before applying expensive operators, and the ability of the robot to tune its vision software parameters during execution, according to what it knows about the difficulty or time-constraints of the problem. For example, one of the visual routines for object search that we have developed (Firby *et al.* 1995) can use a color model of the object to restrict search to areas where it is more likely to be found, prior to edge matching. The edge matcher is restricted by knowledge of the size of the object, and what is known about its likely orientations and locations. As another example, when tracking an object,

the robot lowers the level of resolution at which images are processed to speed processing and thereby improve eye-body or eye-hand coordination. The loss of acuity is acceptable because the improved focus of attention created by tight tracking greatly reduces the number of false matches that might otherwise result from degraded inputs.

We would like to push the use of context much further. For example, as the robot moves closer to an object that it is tracking, it should be able to adjust the level of resolution downward. Also, if the robot needs to do a number of visual tasks at once (e.g., "free-space" obstacle avoidance and target tracking) it could tune each algorithm for improved speed, at the cost of some accuracy. These are two examples of using *task context* to improve visual performance. We can also gain from using *environmental context.* For example, the robot could decide which low-level cues it will use for determining regions of interest, according to feedback from the visual routines about the usefulness of the cues in a previous scene, or perhaps from long-term knowledge or short term memory of what is likely to be around it. The visual interest cues provide *image context* that not only can be used to limit computation to a part of the scene, but that places geometric constraints about how an object can appear if it is to be in a certain region of the image. In general, careful restriction of viewpoint search based on image location may ameliorate the inherent combinatorics of most general purpose model-matching algorithms.

We want to make it possible for a robot to construct, at run time, from a fixed set of visual operators, a visual routine appropriate for the current goal, and adapt it on-the-fly to the state of the task and environment. The routine itself would exploit appropriate low-level cues to reduce the aspects of the scene under consideration.

Gargoyle: Run-time configurable pipelines for active vision

In developing Gargoyle, one of our aims is to provide *composability* that is available in most vision software packages, but not at the overhead required by these packages. For example, Khoros' Cantata is an interpreter that can combine operators into new routines at run-time. Unfortunately Cantata's design is not suited for real-time processing, mainly because it uses expensive inter-process communication to pass copies of images from one module to another. Xvision (Hager & Toyama 1994) allows new operators to be derived very flexibly from others, but requires new C++ classes and recompilation to use them.

Although no vision software package provides the performance and run-time flexibility we seek, we don't want to write an entirely new system if there is one that provides an adequate base. We have chosen the Teleos AVP vision system[1], which provides low-level real-time image processing, because it is designed for high performance, and is supported on a standard Windows NT platform.

Gargoyle is a multithreaded, multiprocessing Java program that calls external C or C++ functions for most image processing. It augments the Teleos AVP vision library to provide the kinds of visual routines an autonomous mobile robot needs for navigation, target recognition and manipulation, and human-computer interaction, and is extendible to other tasks.

Gargoyle provides a run-time interpreter that allows dynamic configuration of image-processing pipelines. The pipelines constitute visual routines for accomplishing specific visual goals. Gargoyle includes a set of visual operator modules that are composed into pipelines, and it can be extended with new modules.

The pipelines are constructed out of linked image-processing and computer vision modules. In a graphical representation of a visual routine in Gargoyle such as is shown in Figure 1, images and regions of interest (ROI) can be thought of as flowing one-way along the links. A feature of the Gargoyle runtime system is that no unnecessary copying is done when executing these pipelines. Images can be tagged with viewer state information that can be used to translate from image to world coordinates.

Gargoyle communicates with a robot control system (we use the RAP reactive plan-execution system (Firby *et al.* 1995)) via string messages. There are command messages sent to Gargoyle for constructing, reconfiguring, and executing pipelines. Gargoyle returns messages that contain the results from pipeline execution, and error conditions from modules within a pipeline.

Data structures and processing modules

Gargoyle defines two basic data structures, images and regions of interest, and provides a set of vision modules that process images within regions of interest. This section describes the image and ROI data structures, and the vision modules we provide.

ROI: Regions of Interest

The **ROI** defines a rectangular area in an image, and specifies a level of resolution as an integer subsampling factor. Optionally, individual pixels with the region may be masked out. ROIs are used to restrict subsequent image processing to the given area. They are

[1] http://www.teleos.com

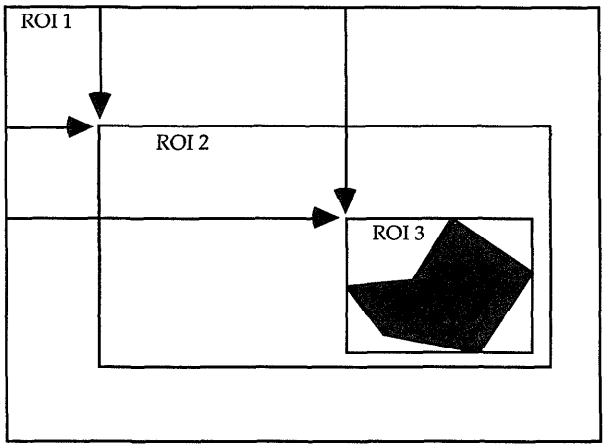

Figure 1: Region and sub-region of interest within full field of view. Offsets defining a region are always with respect to full field of view. Regions are rectangles with optional binary masks.

also used to annotate image buffers with information about the source of the data in the image.

A ROI is specified with respect to a full size, high-resolution image area, whose upper left pixel is indexed at $(0, 0)$ (Figure 1). A region is defined by its offsets, measured in pixels from the upper left corner of the full field of view, and its width and height. Regions of interest are created from other regions by specifying the offset and size of the subregion within the larger region. If the larger region was itself a subregion, *its* offsets are added to the new offsets, so that all ROIs remain in a canonical coordinate system.

ROIs may also include a binary 2-D mask for defining regions that are not rectangular. There are methods to initialize the mask to the entire region or to a pre-defined mask, empty it, and add or remove pixels from it. ROI objects also provide iterators that sequentially return each location in the region, whether it is a simple rectangular region or an arbitrary set of pixels in a mask.

Calibrated regions of interest

Regions of interest can be calibrated with an object that describes the viewer's (i.e., camera's) location and pose in a world coordinate system. The calibrated ROI class is derived from the ROI class and adds camera height, location, viewing angles, magnification, real clock time, and a flag to indicate if the camera is in motion. To get this information, Gargoyle calls a global function called **viewerState**, with a single parameter for the camera number in a multiple camera system,

From	To
monocular image point	3D line of sight
stereo image point pair	3D world location
3D world location	image point
3D world volume or plane	image region

Table 1: Calibration functions

whenever image data is created (see Feature Maps, below). The Gargoyle user must replace this function with one that can query the robot control system for its current location, bearing, and the position and magnification of the camera if these are adjustable. The values are simply stored with the region of interest. Since images also define regions of interest, any image can be annotated with viewer state information.

Gargoyle includes a camera model class that can be parameterized for different cameras. The model can convert from image coordinates into lines of sight and back, and, if an image is annotated with viewer state information, image locations and regions can be converted into real world locations and regions, as listed in table 1. Finally, Gargoyle can generate image ROIs corresponding to volumes in real-world coordinates, using the current viewer state. This is needed in order for the robot to give hints about where the visual system should look for an object. For example, if the task is to clean up trash from the floor, then the visual system only needs to look in the parts of the image that correspond to the floor and some small height above it. If the cameras are tilted down so that only the floor is visible, the ROI generated will include the entire image, but if the camera needs to be raised slightly for some other purpose, or can't be moved, image processing will still take place only where it makes sense for the current task.

Image buffers

Image buffers hold one or three-band images, and are defined as a C++ template class. The class provides methods only for pixel access, reading and writing to disk, and setting a color space tag. A visual processing module is available to convert images from one color space to another.

Image buffers inherit from the calibrated region of interest class, above. The ROI defines the source region of the image with respect to the maximum field of view. It also indicates if the pixel data represents a subsampling of full-resolution data. If the viewer state global function is redefined for the robot, the image will be annotated with the camera's position. If this information is not available, the image is simply

Map	Output	Parameters
Color	3bnd int.	color space
Gray scale	1bnd int.	none
Contrast	1bnd signed	filter size
Edge	1bnd binary	filter size
Motion	1bnd int.	max. movement
Disparity	1bnd int.	max. disp.
Frame diff.	1bnd signed	interval

Table 2: Gargoyle input modules or feature maps. Outputs (1 or 3 band images) are processed as inputs by other modules in a pipeline.

stamped with the time it was taken and the camera number. Images, including viewer state information, can be written to disk, so that real experiments can be replayed later. This makes it possible to debug an experiment any time after it takes place.

Visual processing operators

The processing modules which comprise every visual routine fall into three general categories, based on the type of links into and out of the operators. *Feature maps* provide system input. In a pipeline they take no image input, but have an optional ROI input, and produce an image buffer as output. *Segmenters* take an image and optional ROI input, and produce an ROI output. *Processors* take image and ROI inputs, and may produce an image output. They also may send result messages to the robot control system. The bulk of image processing and computer vision modules are processors in this scheme.

Feature Map Modules Feature maps get input from the cameras. For efficiency and programming simplicity, Gargoyle provides a small set of modules that interface to the underlying image processing system (Teleos AVP). These modules provide the input to all pipelines, and are often used by several visual routines that execute simultaneously. To efficiently share information about features, and to keep inputs to separately executing routines in synchronization, the lowest level of the Gargoyle system is a fixed set of image feature map modules. Since these modules include color and grayscale images, users can perform any type of image processing by starting with these, and adding new processing modules (see below).

The set of feature maps is determined by the underlying AVP hardware and libraries. The Gargoyle user is not expected to add new feature maps. The feature maps available are shown in table 2. Figure 2 shows the contents of an edge feature map.

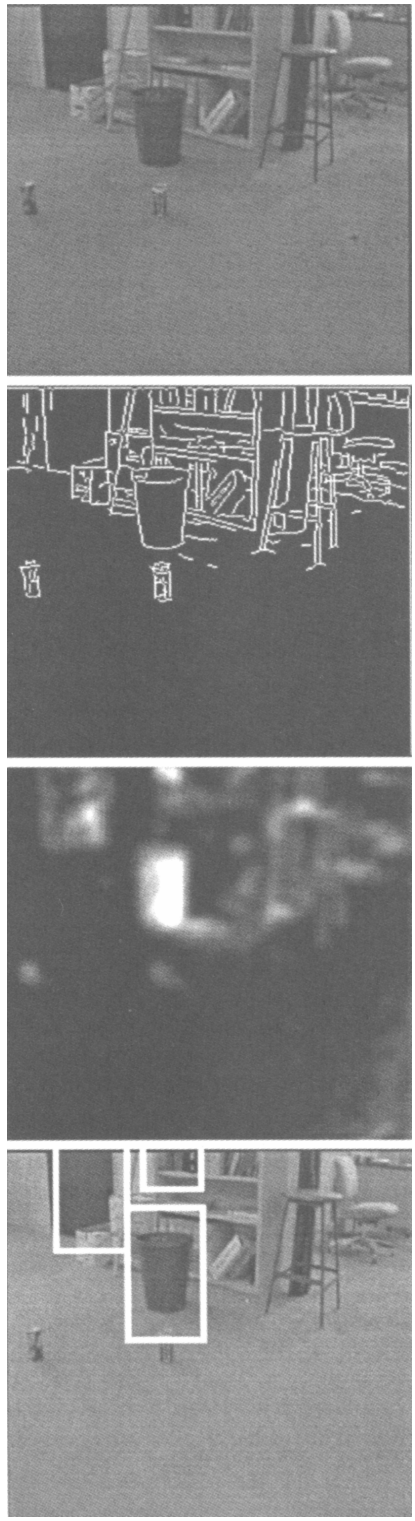

Figure 2: top: gray-scale image; second: binary edge image; third: color histogram back-projection of a picture of a brown trash can, bottom: rectangular regions-of-interest around local peaks

Segmentation and Region of Interest Modules These modules split an image into one or more regions for further processing. The output is a *stream* of ROIs. Some modules may use the optional image mask to define nonrectangular ROIs. Typically, the ROI stream is passed to a cropping module which produces a stream of sub-images corresponding to the ROIs. This image stream is then piped into other modules for processing. The segmentation and ROI-generating modules standard in Gargoyle are listed in table 3.

In figure 2, the local peak segmenter finds local peaks in an intensity image, which in this case is a color histogram back-projection image. The peaks must be separated by a minimum distance, which is a parameter of the module. The peaks are bounded with rectangles whose edges are determined by where the intensity image falls below a threshold fraction of the local peak value, measured along horizontal and vertical lines through the local peak point. The threshold fraction is another parameter of the module.

The connected component segmenter uses image morphology operations to find connected or nearly connected binary image segments of sufficient size. Each connected component generates a rectangular ROI with a binary mask that gives the exact shape of the segment. Subsequent image processing would normally use the ROI pixel iterator to step through only the pixels in the segment. This is how, for example, the average intensity of an arbitrary segment would be computed.

The tracker module tracks the location defined by ROIs as they come into the module, and generates a prediction of the next ROI it will see. Since each ROI is stamped with the time that it was produced (from a feature map, originally), the tracker knows how fast the ROI is moving in image coordinates. It also can compute when the next ROI will likely be produced, because it tracks the time intervals between successive inputs. The output is a predicted ROI which can be used to direct perception toward where a moving object will likely be next.

The world volume ROI module produces an image ROI corresponding to a rectangular volume in the world. The volume is set from the robot control system as parameters of the module. The module does not take any input from the pipeline. This module requires a user supplied viewer state function, and camera parameters (see Calibrated ROIs, above). If the world volume described does not project into image based on the current viewer state, an empty stream of ROIs will be generated.

The set operation module combines two ROIs into one, using the intersection, union, and subtraction operators.

ROI module	Pipe In	Parameters
Local peaks	scalar img	thr, min sep
Conn comp	bin img	max gap, min area
Tracker	ROI	smoothing rate
World vol	calib. ROI	rect vol
ROI comb	ROI a, b	set op ($\cap, \cup, -$)

Table 3: Region of interest and segmentation modules. Each module produces a ROI or stream of ROIs for further processing.

Filter module	Input	Output	Parameters
Threshold	img	bin img	thr
Convolution	img	img	kernel
Warp	img	img	interp func
Frame avg.	img	img	interval
Color conv.	img	img	color space
Back-proj.	3bnd img	img	color hist
Cropper	img	img	none
Lines	img	bin img	thr, min seg

Table 4: Image filtering modules. Outputs from these modules are images that are processed as pipeline inputs by other modules.

Processing modules The processing modules comprise the bulk of the Gargoyle system. There are filtering modules for processing images into other images, and recognition and measurement modules for finding objects or calculating properties of the scene. The filter modules currently in use are shown in table 4. These are largely self-explanatory. The color back-projection module produces an intensity image whose pixel values indicate the "saliency" of the corresponding color image's pixel as an indicator of the presence of a known colored object (Swain & Ballard 1991). New filters are easily added using a C++ template class.

Recognition and measurement modules implement computer vision algorithms, which can be very complex. In keeping with our research strategy we have tried to use well-established algorithms and software when possible. The modules we have incorporated so far are listed in table 5. These modules normally produce results that need to be returned to the robot control system, rather than an image that is sent further along the pipeline. Table 5 shows the messages that are sent back to the control system.

New algorithms can be added using a template class. The programmer has access to an input image with a pixel-access iterator that will step through the image.

Module	Input	Signal	Params
templ match	bin img	target loc	model, thr
pat match	bin img	target loc	model, thr
free-space	bin img	ranges	resolution

Table 5: Computer vision modules. The outputs of these modules encode answers to perceptual queries and are transmitted as signals to the RAP system

If the image is calibrated with viewer state information, the algorithm can use image point-to-world-line functions (table 1) to get information about where the image points may come from. For example, we find the lines of sight corresponding to the highest and lowest parts of the image, and intersect those lines with the plane of the floor to determine how far away an object in the image could be if it is assumed to be on the floor. This is used to control the viewpoint search in the Hausdorff template matching algorithm.

Visual routines as pipelined processes

Consider a visual routine for finding objects that we use frequently in our research. This routine searches for an object by shape, matching a library of edge models against an edge image. Other binary models and images could be used instead of edges. The model database contains different views of the object, which are needed to the extent that the object's surface is not planar or circularly symmetric. The search algorithm (Huttenlocher & Rucklidge 1992) translates, scales, and skews the models as it searches through the space of possible viewpoints. The viewpoint space can be controlled through parameters depending on the task context and knowledge of where the object might be, as well as through calibrated image context as described above.

Searching across a large space of viewpoints is expensive; with a cluttered, high resolution (roughly .25 megapixel) image, searching from all frontal viewpoints between, for example, 0.5 to 5 meters, for a single model, takes over a minute on Sun Sparcstation 20. To reduce the space of possible viewpoints, we restrict the search to areas of the scene containing colors found in the model. This requires a color model of the object, which is represented as a 256 bin histogram of an 8-bit color-resolution image of the object. The 8-bits could represent, for example, 4 bits of hue, 4 bits of saturation, and 0 bits of value (a distribution we have found effective).

Figure 3 shows the processing pipeline that implements the routine. Two feature maps generate input simultaneously, or as close together as the underlying

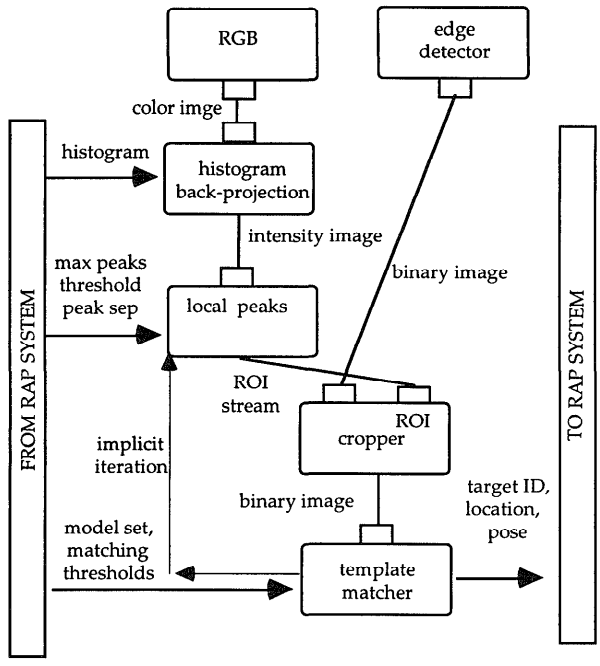

Figure 3: An image processing pipeline for finding objects by shape, using color as an interest cue.

hardware allows. The RGB map is piped into a color histogram back-projection module. The RGB map can be parameterized with a low-resolution (subsampled) ROI so that the output image is very small and later segmentation is fast. The edge detector can also be operated at different levels of resolution according to how large and nearby the object is expected to be. The modules need not operate at the same resolution.

The color histogram back-projection output is piped into the local peak segmenter to generate regions of interest where the object could possibly be found. The output of the local peak module is a stream of ROIs. The interpreter processes the stream by marking the module as the head of an *implicit iteration loop*. Each ROI in the stream is then processed separately by the next module. When the pipeline bottoms out, that is, no more links exist, or a module finishes processing without generating an output image, the interpreter returns to the nearest upstream iteration head, and continues processing with the next ROI. After the last ROI is processed, a special end-of-stream ROI is sent to the next module and the interpreter notes not to return for processing. Most modules do nothing but pass the end-of-stream markers on. Others may process a stream of ROIs and wait for the end-of-stream indicator to perform a calculation based on the entire

stream of inputs. The hausdorff template matcher is one such module, as we will see.

The ROI stream from the local peak segmenter passes into the cropper, which also receives an image input from the edge detector. The cropper produces a binary subimage from the edge image for each ROI passed in. The resolution of the binary image is preserved, even though the ROI may be at a lower resolution.

The template matcher searches each image for instances of its models. The viewpoint search is controlled according to parameters from the robot control system, which may know how far away the object should be, and by inferring the possible viewpoint of the object if it is to appear in the image, as described earlier. The template matcher runs in several search modes, in that it can report an answer as soon as it finds any match, or wait until the stream of images has been processed to return the best match. When the robot receives an answer from the module, it can tell the interpreter to halt further execution of the pipeline.

Toward more flexible visual routines

Pipeline configurations and the parameters for their operators can be stored as a library of visual methods appropriate for different sensing tasks and environmental conditions. We use the RAP system to store the configuration commands for constructing pipelines. Visual routines, then, are represented as RAP plans for achieving particular perceptual goals.

On-the-fly pipeline reconfiguration

One somewhat more flexible routine we are developing uses two low-level cues, motion and color, for following a person (Prokopowicz, Swain, & Kahn 1994). A RAP encodes the connections for a simple pipeline to track an object by motion or color. The encoding is sent to GARGOYLE to build the pipeline. The pipeline is very simple to begin with: a motion map is piped into the local peak segmenter, which finds sufficiently large areas of motion. These are piped into a ROI tracker module. A color feature map is also created and piped into a histogram backprojector. This feature map is initially turned off to save processing power. GARGOYLE will not execute a pipeline whose input is off. The RAP is sketched in Figure 4.

If more than one moving area is segmented, the tracker module will follow the largest one, but will also send a warning message (:no-target or :multiple-target), which the GARGOYLE interpreter sends to the RAP system. Likewise, if no region is found, a warning is sent. The RAP level is where the knowledge for how to track an object under different conditions is encoded. In this case, the RAP responds to the warning by instructing the GARGOYLE interpreter to reconfigure the pipeline so that a different module is used as input to the segmenter. If the RAP used the motion map the last time, it will use color and histogram backprojection. Of course, flipping between two modes of segmentation is only the simplest example of run-time flexibility. We feel that GARGOYLE provides the basis for flexible real-time vision, and that a high-level planning system like RAPs provides the reasoning and knowledge representation capacity needed to exploit that flexibility in much more interesting and powerful ways.

Because Gargoyle is a single process, creating and linking a module into a pipeline only involves adding an entry to the current configuration graph in memory. Executing a module requires spawning a new thread,

```
(define-rap
 (track-motion-or-color ?target-name ?color-model)

 (init (pipeline-create 1
    (1 motion-feature-map RESLEVEL1)
    (2 local-peak-segment MINPEAK MINSEP)
    (3 ROI-tracker)
    (4 RGB-feature-map)
    (5 histogram-BP ?color-model RESLEVEL2)
    (1 into 2)
    (2 into 3)
    (4 into 5)
    (4 off)
    (1 off)))

// track by motion
 (method (context (not track-by color))
    (pipeline-config 1 (1 into 2) (1 on))
    (pipeline-execute 1)
    (wait-for (:object-at ?x ?y)
      (mem-set target-at ?target-name ?x ?y))
    (wait-for (or (:no-target) (:multiple-target))
      (pipeline-config 1 (1 off))    // stop pipe
      (mem-del track-by motion)      // use color
      (mem-add track-by color)))     // next time

       // track by color
 (method (context (track-by color))
   (pipeline-config 1 (5 into 2) (4 on))
   (pipeline-execute 1)
   (wait-for (:object-at ?x ?y)
     (mem-set target-at ?target-name ?x ?y))
   (wait-for (or (:no-target) (:multiple-target))
   (pipeline-config 1 (4 off))       // stop pipe
     (mem-del track-by color)        // use motion
     (mem-add track-by motion))))    // next time
```

Figure 4: RAP that constructs and modifies visual routine for flexible tracking using color and motion cues.

which is also a very fast operation. The interpreter runs as a separate thread and accepts messages from the skill processes asynchronously, so it can respond immediately to requests to reconfigure, reparameterize, or control execution, even during image processing.

Related work

Ullman proposed that visual routines (Ullman 1984) which answer specific perceptual questions concerning shape and geometric relationships can be constructed out of elemental visual operators and control structures. Our work is fundamentally similar, but explores the issues that come up when these ideas are extended to a real-time active system that is interacting with its environment. In particular, it emphasizes the need for different ways to compute the same result, depending on the immediate context.

The Perseus vision system (Kahn & Swain 1995) is being developed at the University of Chicago to augment human-computer interaction through gestures such as pointing. Perseus is currently implemented with the DataCube server as the underlying visual processor. Perseus is based on the notion of active visual objects, or markers, that are spawned to recognize and track relevant parts of a scene, such as a person's head and hands. One of the Gargoyle design goals was to provide a platform to support Perseus on a visual robot. The markers will be implemented as RAPs that construct image pipelines for recognition and tracking. Gesture recognition is carried out by a higher-level visual routine that spawns markers and monitors their locations, looking for certain configurations that indicate human gestures.

Horswill has shown that it is possible to translate a Horn clause into a custom program, written in a "visual-computer assembly language", that attempts to satisfy the clause. The operators of this assembly language are much more primitive than what most robot researchers would like to use, and the underlying hardware (the Polly robot) is not in widespread use. Gargoyle will provide ourselves and other robotic researchers with the means for writing visual programs in a high-level language, using the best computer vision algorithms and software as operators, on standard PC hardware.

Conclusion

Gargoyle will provide us with a tool that is sufficiently flexible to create much more specialized and efficient visual routines that are adapted to solve specific tasks, and fully exploit run-time context, with no significant cost in run-time efficiency. Because Gargoyle will run on a standard, multi-platform operating system, it can be easily and affordably ported to other systems, and will benefit from further advances in these platforms. Gargoyle's extendibility will allow it to be useful for other domains besides the mobile robot domain described here – we are already working towards its use in creating a human-computer interface for a virtual reality environment.

References

Aloimonos, J. 1990. Purposive and qualitative active vision. In *International Conference on Pattern Recognition*, 346–360.

Bajcsy, R. 1988. Active perception. *Proceedings of the IEEE* 76:996–1005.

Ballard, D. H. 1991. Animate vision. *Artificial Intelligence* 48:57–86.

Chapman, D. 1991. *Vision, Instruction, and Action.* MIT Press.

Firby, R. J.; Kahn, R. E.; Prokopowicz, P. N.; and Swain, M. J. 1995. An architecture for vision and action. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Hager, G. D., and Toyama, K. 1994. A framework for real-time window-based tracking using off-the-shelf hardware. Technical Report 0.95 Alpha, Yale University Computer Science Dept.

Huttenlocher, D. P., and Rucklidge, W. J. 1992. A multi-resolution technique for comparing images using the hausdorff distance. Technical Report CUCS TR 92-1321, Department of Computer Science, Cornell University.

Kahn, R. E., and Swain, M. J. 1995. Understanding people pointing: The Perseus system. In *Proceedings of the IEEE International Symposium on Computer Vision.*

Prokopowicz, P. N.; Swain, M. J.; and Kahn, R. E. 1994. Task and environment-sensitive tracking. In *Proceedings of the IAPR/IEEE Workshop on Visual Behaviors.*

Swain, M. J., and Ballard, D. H. 1991. Color indexing. *International Journal of Computer Vision* 7:11–32.

Ullman, S. 1984. Visual routines. *Cognition* 18:97–159.