In *Proceedings of the Fifth International Conference on Cognitive Modeling*, 189-194. Bamberg, Germany: Universitats-Verlag Bamberg.

Connecting a Cognitive Model to Dynamic Gaming Environments: Architectural and Image Processing Issues

Kunal Shah (kdshah@eos.ncsu.edu)
Sameer Rajyaguru (srrajyag@eos.ncsu.edu)
Robert St. Amant (stamant@csc.ncsu.edu)
Department of Computer Science, North Carolina State University
Raleigh, NC 27695 USA

Frank E. Ritter (ritter@ist.psu.edu)
School of Information Sciences and Technology, The Pennsylvania State University
University Park, PA 16081 USA

Abstract

This paper describes an image processing system that has been coupled with the ACT-R cognitive modeling architecture. The system supports interaction with dynamically changing visual environments associated with off-the-shelf computer games running independently of the model. The image processing techniques have been applied to three different games and are intended to be extensible to others. This paper discusses the image processing approach and its strengths and limitations. The implications for cognitive modeling are twofold: the image processing system can be used across different models, extending their range, and it provides a closer integration with more naturalistic environments.

Introduction

A basic concern in cognitive modeling is the representativeness of problem-solving environments. Ideally, we want environments that the model lives in to represent all of the relevant details that humans are aware of and constrained by in carrying out a task. The most direct way of satisfying this goal is to build models that can interact with the real world; however, we find a wide gap between the input capabilities of existing cognitive modeling architectures and the information that natural environments commonly make available.

Recent research efforts have adopted the more modest goal of building cognitive models that can interact with software environments designed for human users. Everyday computer productivity applications contain text, numbers, and discrete objects and symbols in relatively simple arrangements; these environmental properties make it feasible, sometimes even straightforward, to accommodate the input requirements of a symbolic cognitive model.

Significant progress has been made through the integration of cognitive models with software environments, as demonstrated by the growing literature of ACT-R research (Anderson and Lebiere, 2000). Still, these environments are relatively simple in comparison with natural environments. They tend to be static, discrete, and predictable, properties that can be exploited by a model but that simultaneously limit the range of results that can be reached in experimenting with them.

We are attempting to overcome this limitation, by building models that can interact with computer-based video games. Games have played an important role in helping cognitive modelers gain insight into the process of human reasoning. Historically, strategy games such as tic-tac-toe and chess have led to an improved understanding of human cognition (Newell and Simon, 1972). More recently, dynamic games have attracted attention as testbeds in which dynamic real-time human decision-making can be observed and reproduced.

Visual processing and analysis are key to effective human behavior in these environments, but this aspect has been neglected in cognitive modeling research on computer games. Cognitive models are most often built to communicate with the application programming interface (API) to software environments, bypassing the complexities of image processing. We believe that eventually, if we are to reach the goal of building models that automatically interact with a wide range of real environments, the issue of visual processing must be addressed. Our work takes early steps toward the goal of integrating cognitive models with environments.

In this paper we describe the properties of visual environments, with a focus on computer games, that are relevant for the design of an image processing component in a cognitive model. We describe our image processing architecture, which has been adapted to three different game interfaces. Our efforts have mostly concentrated on one of these games, a driving simulation. We explain how the image processing system has been extended from a set of general-purpose techniques to include functions specific to the driving game, to support a realistic model of human driving.

We believe that our work has implications for cognitive modeling in games (Laird, 1999), models for robot agents, and models for user interface evaluation (Ritter and Young, 2001).

Visual environments for cognitive modeling

The design of an image processing component for a cognitive model is not solely dependent on the model; the environment and the tasks to be carried out are also important factors. Designers must consider the efficiency, robustness and accuracy of candidate image processing algorithms and tradeoffs with the requirements of the cognitive model. We can summarize environment properties (and to some extent task properties) as follows.

Static versus dynamic environments. In some environments, changes take place only in response to the actions of the model. In a gaming environment, monitoring and real-time responses in the image processing component are necessary for the model to maintain an accurate representation of its properties.

Discrete versus continuous environments. A environment is effectively continuous if it is characterized by patterns that vary over a range of values much greater than can be individually accounted for symbolically (e.g., arbitrary numerical values, hues, or auditory signals.) Digitized environments, such as the pixels of the screen image of a game, are effectively continuous if the individual pixel values and relationships are not meaningful to the cognitive model. The goal of image processing is to translate continuous attributes into discrete values that can be handled by the model.

Predictable versus unpredictable environments. In some environments, it is possible to predict the next state from the current state. Static environments have high degree of predictability, though this may change when actions are initiated. The games we have considered in our research are also to some extent predictable. For example, if objects always move in straight or at least continuous trajectories, then once an object's visual representation has been processed it can be tracked instead of iteratively reprocessed.

"Simple" versus "complex" environments. The complexity of the objects constituting the environment is a dominant factor in the design of an image processing system. There are several dimensions to complexity:

- *Shape* is generally the most relevant cue for object recognition. If the shapes to be recognized are known in advance, matching can be used. After preprocessing, the image is partitioned into distinct regions, which are assembled combinatorially to form super regions. These can then be matched against prototypes of possible objects. Arbitrary shape recognition is much more complex, and in our work can only be dealt with by specialized functions.
- *Color* and *texture* are other important cues for object recognition. Normalization and quantization, combined with edge detection for contour analysis, and texture analysis can be used to segment objects based on color and texture.
- *Motion* can also play a role in detecting objects. The best example of this is in camouflaging, when it is not possible to distinguish the bounds of an object based on color or shape. Many image analysis techniques use motion information as the basis for segmenting objects from background. In some games attending to motion can be the most efficient way of focusing attention on properties of the environment that are relevant at the current time.
- *Spatial relationships* can constrain the amount of visual information that needs to be processed. This naturally affects processing time.
- *Spatial positioning* is also relevant. In some environments, only a subset of the visible scene may need to be processed, rather than the entire visible region. Further simplification is possible if the subset occurs at a fixed position.

Sparse versus crowded environments. A final factor is the number of objects to be processed. The complexity of objects of interest, as discussed above, is generally the most important factor. However, if only a few objects need to be considered, whether simple or complex, then the burden on the image processing algorithm is reduced dramatically and the requirements of fast computation and efficiency are relaxed.

In our past work on visual processing for cognitive models, we have concentrated on environments that are static, predictable, simple, and relatively sparse. The focus of our past work has been to translate effectively continuous patterns, represented on the screen at the pixel level, into symbolic representations of characters, widgets, and other standard visual objects in the Windows environment. The games we discuss in the next section are dynamic, less predictable, and more complex in a variety of ways. They are still relatively sparse and do not differ qualitatively from standard productivity applications with respect to discreteness and continuity.

Game environments

We have worked with three different games, whose interfaces are shown in Figures 1, 2, and 3. All of these games were developed by others and have not been modified by us. Figure 1 shows a first-person driving game, in which the model controls the speed and steering of the car. Figure 2 shows a Mars rover simulation. The goal is to direct the rover over the planet surface, collecting specimens of the local fauna.

When the rover collides with rocks, these disappear and release a small swarm of creatures to be chased and captured. Figure 3 shows a Mars base exploration game. This game is considerably more complex than the others, requiring users to reason about resources, objects with different capabilities, autonomous agents, and spatial relationships. Although we have developed image processing functions that can parse all three of these environments, the only environment for which we have constructed complete cognitive models is the driving game.[1] This game will thus be the focus of our discussion below.

The models for the driving game are based on the ACT-R architecture. The 5.0 release of ACT-R interacts with user interfaces using a Perceptual-Motor component (ACT-R/PM). ACT-R/PM (Byrne, 2001) includes tools for creating and modifying user interfaces so that models can see and interact with interface objects via hooks provided in the development programming environment. This allows most models to interact in some way with most interfaces that are written in the underlying ACT-R development language, Common Lisp, and to let all models interact with all interfaces written with the special tools.

We have developed a more general version of ACT-R/PM, which provides ACT-R direct access to an interface, removing the need for a specific interface creation tool. The extension, which we call SegMan, takes pixel-level input from the screen (i.e., the screen bitmap), runs the bitmap through image processing algorithms, and builds a structured representation of the screen (St. Amant & Riedl, 2001). SegMan can also generate mouse and keyboard inputs to manipulate objects on the screen. This functionality is called through the ACT-R/PM theory of motor output, but we have extended the output results to work with any Windows interface. This is done by creating very primitive events (click icon, select button, etc.), which are implemented as functions at the operating system level. As such, they are indistinguishable from user-generated events. Currently, we have a fully functional system that runs under Windows 98 and 2000.

In the next section we describe the architecture of the image processing component, its design rationale, and its application to the driving game.

Image processing in SegMan

Computer vision and image processing are two distinct but closely related fields falling under the umbrella of computer imaging. This distinction is based upon who is the ultimate receiver of the visual information. Image processing algorithms generate results that are used by people in a variety of different

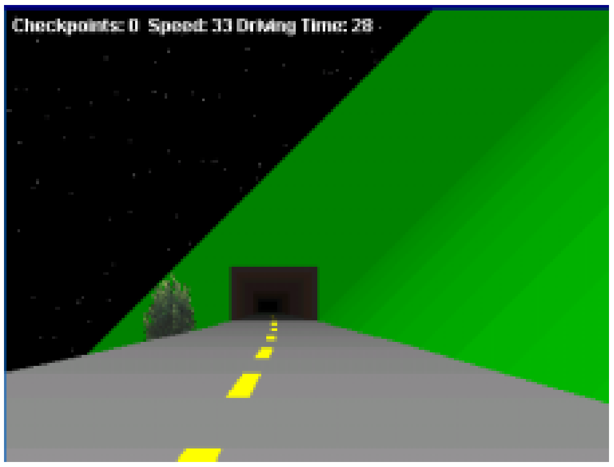


Figure 1: Driving environment, available at www.theebest.com/games/3ddriver/3ddriver.shtml.

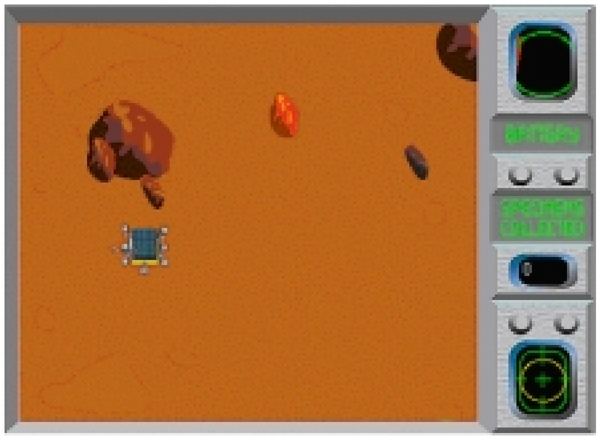


Figure 2: Mars rover environment.

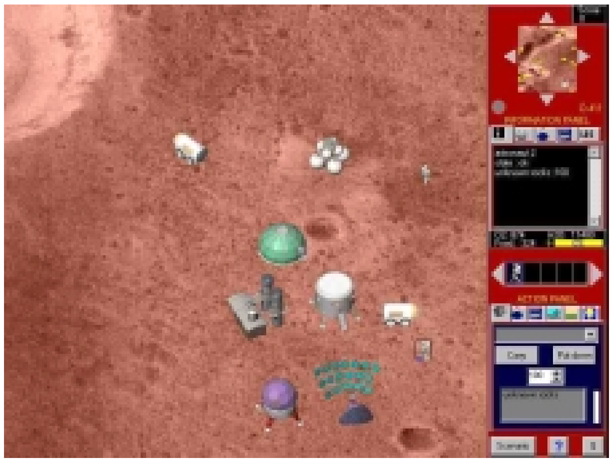


Figure 3: Mars base environment, available at perso.wanadoo.fr/salotti/marsbase.htm.

[1] Information on visual processing in the model, how it influences cognition, and how the code module can be reused, plus details on the model s knowledge and behavioral predictors, can be found in the following unpublished paper: www4.ncsu.edu/~stamant/papers/HRI-v406-25.pdf.

domains; computer vision applications generally produce results that are targeted for further automated processing, for example by an AI agent or, in our case, a cognitive model (Umbaugh, 1998). Image analysis, combined with feature extraction and pattern classification, is the key to a computer vision system, the end product being the extraction of high level information (e.g. objects) from an image. Most techniques used for image processing can also be found in computer vision systems.

As developers we face a practical tradeoff between building models that respect the known properties of biological vision (taking the computer vision approach) and building models that can run fast enough on conventional computer hardware to interact with off-the-shelf applications in real-time (taking the image processing approach.) We have leaned toward the latter, with the hope that future research will explore the former direction (e.g., see (Chapman, 1992)), as our computational resources increase and our software improves.

The fundamental concept underlying any visual-processing algorithm is a process called segmentation. Segmentation is the process of delineating regions that constitute an object and separating them from the background in an image. In our domain, is directly influenced by the category to which the game belongs. The object recognition process (as a part of image analysis) can be viewed as a sequence of preprocessing of the image, data reduction and morphological filtering, and feature extraction and analysis. In our discussion of these steps, we mention the existence of alternative techniques for accomplishing different goals; a detailed discussion of such techniques, even to the extent of giving examples, is beyond the scope of this paper. (See instead Umbaugh (1998).)

Preprocessing. During this stage, the image may be quantized (reducing the number of color levels or spatially) or it may be enhanced to prepare it for the subsequent processing steps. Some other image geometry operations such as cropping, zooming, shrinking, enlarging, translating, and rotating may be performed on the entire image or parts of image (these parts are called regions of interest). Another important and widely used preprocessing operation is edge detection, for which various different techniques can be applied. Once the edges have been detected, it is often necessary to find lines. A line may be defined as a collection of edge points that are adjacent and have the same direction. Hough transforms are designed specifically for this purpose. All these operations prepare the image so as to make the data reduction and feature extractions tasks easier.

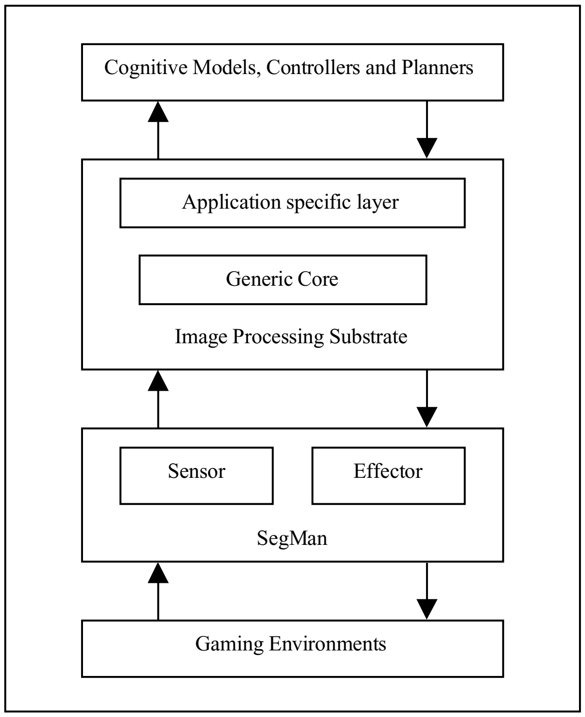
Data reduction and morphological filtering. Data reduction algorithms take the preprocessed image and reduce the image data such that it can be analyzed by feature analysis algorithms. This is a crucial step in solving the object recognition problem, and image segmentation is the key to it. Morphological filtering refers to changing the structure or form of the image. As stated above, the goal of image segmentation is to divide the image into regions, which may represent an object in its entirety or may be part of a larger object. Various methods exist to segment an image into regions with varying levels of complexity and the accuracy with which the image is segmented. The basic idea underlying these methods is that the objects can be distinguished by either considering them to be a lump of pixels with some measure of homogeneity in terms of features such as color, brightness, and texture, or perceiving them as contrasting with other objects on their borders. Another important issue is related to connectivity between segments, i.e., deciding which segments should be combined to represent an object.

Most image segmentation algorithms make use of region growing and shrinking, clustering, and boundary detection, alone or in combination. Each of these areas encompasses a variety of methods.

Feature Extraction and Analysis. Feature extraction can be viewed as an extension of the data reduction process. Once the features have been extracted, the next step is to classify them to identify objects. This requires application level knowledge and hence application specific knowledge is used in this final phase.

One way to classify objects is to define a feature space and then compare the object's feature vector against the template object's feature vector. A feature vector is an n-dimensional vector such that each dimension represents exactly one feature of the object. Thus, in a simple example, we might represent an object in terms of its average RGB values and its area, giving a 4-dimensional feature space. Different methods are used to compare the similarity (or the difference) between two feature vectors. One of the simplest metrics for measuring the distance between two vectors is Euclidean distance, but other weighted metrics are common as well.

Care must be taken while selecting the features, so as to ensure that the features chosen are robust. For example, if a feature is RST invariant, it will remain the same despite the object being subjected to rotation, scaling or translation. In order to extract features, the image that results from data reduction and morphological transformation is analyzed and labels are then assigned to the objects. The labeled object now can be thought of as a binary image having a value of 1 and the rest of the image is having a value of zero. This image is then used for the extraction of features of interest such as area, center of area, axis of least second moment, perimeter, Euler number, projections, thinness ration and aspect ratio. While the first four are used more commonly and help identifying the location of the object, the latter four are used under specific, domain-



Cognitive Models, Controllers and Planners
Application specific layer
Generic Core
Image Processing Substrate
Sensor
Effector
SegMan
Gaming Environments

Figure 4. System architecture.

dependent conditions and tell something about the shape of the object.

Based on this stepwise decomposition, we have developed a set of functions to act as a core, application-independent image processing component for cognitive modeling, as described in the next section.

The System

Architecture. Figure 4 shows the architecture of the entire system. As shown in the figure, the image processing substrate interacts with the game environment by capturing snapshots of it at regular intervals. For this, it makes use of APIs provided by the SegMan system. Thus, SegMan provides sensor and the effector services to the system. The image processing substrate consists of two layers: a generic core and an application specific layer.

Generic core. The generic core performs functions that fall into the preprocessing stage of the object recognition process. It works on a captured image that is a snapshot of the gaming environment. The following operations are performed on the captured image, as illustrated in Figure 5.

- *Normalize and Quantize Image.* Usually the captured image contains a level of detail (in terms of number of values in the R, G and B streams in the image) that may not be needed to serve the model's purpose of controlling the game effectively. This function normalizes the number

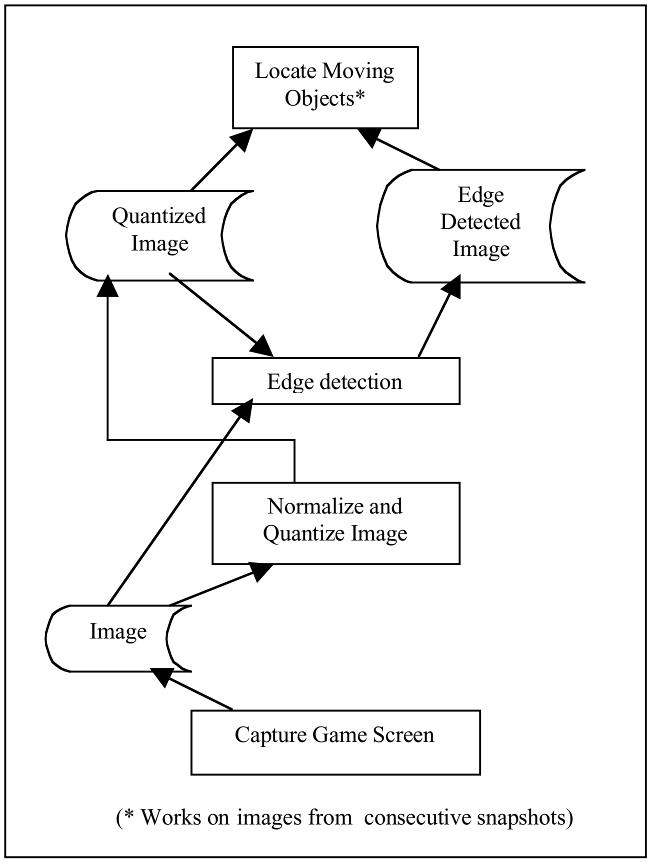


Figure 5: Generic core of image processing substrate.

of color levels per stream used in the image to a value appropriate to serve the model's needs.
- *Edge Detect Image.* This function highlights changes in color intensity values in the image. A Laplacian 3x3 edge detection kernel is used for convolution.
- *Locate Moving Objects.* This function detects moving objects in successive game screen snapshots. It assumes that the background remains static and that the only change in the environment is due to moving objects. Two consecutive snapshots of the environment are XORed to yield an image that contains only the moving objects.

Application specific layer. This layer performs functions that fall into the feature extraction and analysis stage of the object recognition process. It consists of functions that provide a general level of information about the image. Because tasks inevitably have domain-specific properties, we must tailor the image processing component by adding functions for specific games. For the driving game, the extensions are based on studies of human driving. Studies of driving behavior by Land and Lee (1994) and Land and Horwood (1995) describe a "double model" of steering, in which a region of the visual field relatively far away from the driver (about 4 degrees below the horizon) provides information about road curvature, while a closer region (7 degrees below the horizon) provides position-in-lane information. Attention to the visual field at an intermediate distance, 5.5 degrees below the horizon, provides a balance of this information, resulting in the best performance.

To generate the relevant information, several additional functions are needed, built on the generic core.

- *Get Strips.* This function gives the location of the center of the road to assist the model in maintaining its location in the desired lane.
- *Get Horizon.* This function gives the point beyond which the road cannot be seen, obtained by a linear bottom-to-top traversal of scan lines.
- *Get Left (Right) Road Edge Start.* The road image begins at the bottom of the screen, in a perspective view, bounded on both sides by the edges of the game window. This function returns the points on either side at which the edge of the road is visible and not clipped by the window.
- *Get Left (Right) Road Edge End.* This function returns the analogous points at which the road disappears at the horizon or disappears in a curve around a mountain.
- *Get Left (Right) Slope.* This function provides a mechanism for the model to estimate the degree of curvature that the road is taking.
- *Draw Strip.* This function constructs a continuous strip by interpolating the slopes of the missing parts of the strips from the slopes of the existing ones.

Implementation

The system was implemented in C++ using already existing APIs for interfacing with the windowing environment. To illustrate the performance of the system, a set of 20 runs was performed on a Dell P4 (1,700 MHz, 512 MB RAM) running Windows 2000. Mean processing times over a representative set of images in the driving task are as follows. These numbers constrain the minimum cycle time. A good deal of the computation for these values is shared between the functions and can be cached across function calls, however, making the average cycle time somewhere between one and two seconds.

Function	**Mean**	**Std. Dev.**
Get Field Bounds	1155 ms	31 ms
Get Road Edge (left, right)	1155 ms	31 ms
Get Slope (left, right)	1170 ms	34 ms
Get Horizon	1211 ms	35 ms

Obviously these times do not accurately reflect human visual processing speeds; nevertheless they are fast enough to allow interactive control of the driving game, to support modeling at a level of abstraction higher than the perceptual.

Discussion

Our goal in building these systems is to provide cognitive modelers with a wider range of environments against which their theories can be tested. This work is preliminary in the sense that our system has played a part in the evaluation of only one such environment, which means that its generality remains an open issue. Nevertheless it acts as a useful proof of concept. Our current work is toward building an application-specific layer for much more demanding environments, such as shown in Figure 3, that will exercise all of the capabilities of a cognitive model, from motor actions and perception to reasoning and learning.

Acknowledgments

This project is sponsored by the National Science Foundation, award IIS-0083281, by the Office of Navy Research, contract N0001495RF55555, and by the Space and Naval Warfare Systems Center, San Diego. The views in this paper do not necessarily reflect the position or the policies of the U.S. Government, and no official endorsement should be inferred.

References

Byrne, M. D. (2001).ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human-Computer Studies, 55*(1):41-84.

Chapman, D. (1992). Intermediate Vision: Architecture, Implementation, and Use. *Cognitive Science, 16*(4): 491-537.

Laird, J. (1999). It knows what you're going to do: Adding anticipation to a Quakebot. *Working Notes of the AAAI Spring Symposium on AI and Interactive Entertainment*, pp. 41-50.

Land, M. F., and Horwood, J. (1995). Which parts of the road guide steering? *Nature, 377*:339-340.

Land, M. F., and Lee, D. N. (1994). Where we look when we steer. *Nature, 369*:742-744.

Newell, A., and Simon, H. (1972). *Human problem solving.* Prentice Hall.

Ritter, F. E., and Young, R. M. (2001). Embodied models as simulated users: Introduction to this special issue on using cognitive models to improve interface design. *International Journal of Human-Computer Studies, 55*(1):1-14.

St. Amant, R., and Riedl, M. O. (2001). A perception/ action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies, 55*(1):15-39.

Umbaugh, S. (1998). *Computer Vision and Image Processing.* Prentice Hall.