A Real-Time Vision Module for Interactive Perceptual Agents

Bruce A. Maxwell, Nathaniel Fairfield, Nikolas Johnson, Pukar Malla,
Paul Dickson, and Suor Kim

Swarthmore College
500 College Ave.
Swarthmore, PA 19081
maxwell@swarthmore.edu

Abstract. Interactive robotics demands real-time visual information about the environment. Real time vision processing, however, places a heavy load on the robot's limited resources, and must accommodate other processes such as speech recognition, animated face displays, communication with other robots, navigation and control. For our entries in the 2000 American Association for Artificial Intelligence robot contest, we developed a vision module capable of providing real-time information about ten or more operators while maintaining at least a 20Hz frame rate and leaving sufficient processor time for the robot's other capabilities. The vision module uses a probabilistic scheduling algorithm to ensure both timely information flow and a fast frame capture. The vision module makes its information available to other modules in the robot architecture through a shared memory structure. The information provided by the vision module includes the operator information along with a confidence measure and a time stamp. Because of this design, our robots are able to react in a timely manner to a wide variety of visual events.

1 Introduction

This year Swarthmore College entered robots in two events--Hors d'Oeuvres Anyone? [HA] and Urban Search and Rescue [USR]--at the American Association for Artificial Intelligence [AAAI] robot competition. The Hors d'Oeuvres Anyone? event requires the robot(s) to serve food to the conference attendees at the main conference reception with over 500 people in attendance. The robots are evaluated based upon their ability to cover the room, manipulate food, and interact with the people they are serving. The robot-human interaction is weighted most heavily of the three.

The USR event requires the robot(s) to explore a test arena designed by the National Institute of Standards and Technology [NIST]. The test arena simulates the inside of a building after some type of disaster situation and contains three sections of increasing difficulty. The simplest level is navigable by a wheeled mobile robot, the other two sections require a tracked or 4-wheel drive robot. Robots are evaluated on their ability to explore the arena, detect humans--simulated by motion, heat, and skin-colored mannequins--and exit the area within the allotted time (25 minutes).

B. Schiele and G. Sagerer (Eds.): ICVS 2001, LNCS 2095, pp. 190–200, 2001.

Both events benefit greatly from the use of visual input. The HA event, in particular, requires the robot to sense and react to people and various related features, such as colored conference badges in a timely and appropriate manner. The more information the visual sensing can provide, the more interactive and responsive the robot can be to the people it is serving. In the USR event, visual sensing gives the robots one method of detecting and labeling important features in the environment. In some cases it is the only sensing modality able to reliably detect the simulated people.

Rather than have custom vision processing and robot architectures for each event/ platform, we used a single software architecture for all of our intelligent agents, with different controlling modules for the different events. Both the general architecture and the vision module were shared between three different platforms--a Nomad Super Scout II mobile robot, a Magellan Pro mobile robot, and a workstation that acted as the Maitre'd for the HA event. All three participated in the HA event; the Magellan Pro worked by itself in the USR event.

The robots work under significant processing constraints. The Nomad uses a 266MHz Pentium MMX processor, and the Magellan uses a 350MHz Pentium II. For the HA task, the Nomad's capabilities include speech recognition, speech synthesis, visual processing, a sliding arm controller, navigation, control, facial animation, and inter-robot communication. The Magellan possesses similar capabilities but does not attempt speech recognition or possess a moving arm. The workstation is better equipped with dual 400MHz Pentium II's, but gets a correspondingly heavier load with real-time facial animation and synchronization of the lips with speech.

Because of the wide array of capabilities demanding processor time, the vision module has to avoid monopolizing the processor while still capturing video in real-time and providing timely information. The combination of our overall architecture and the vision module achieves this goal. In part because of this design, our robots won both events as well as the Ben Wegbrait award for the integration of AI technologies at the 2000 AAAI robot contest. The remainder of this paper gives a brief overview of the overall architecture and a detailed description of the vision module.

2 REAPER: An Intelligent Agent Architecture

The overall system architecture--hereafter referred to as REAPER [REflexive Architecture for PErceptual Robotics]--is based on a set of modules. The purpose of each module is to handle one specific task, which include: sensing, reflexes, control, communication, and debugging. The fundamental concept behind REAPER is that the central control module--whether it is a state machine or other mechanism--does not want a flood of sensory data. Nor does it want to have to make low-level decisions like how fast to turn each wheel. At the same time it needs real-time updates of symbolic information indicating what the world around it is doing. The sensor and reflex modules gather and filter information, handling all of the preprocessing and generating intermediate actions required to achieve high-level commands or goals. This is similar to the way our brain seems to deal with a request to pick up an object. While we consciously think about picking up the object, our reflexes deal with actually moving our hand to the proper location and grasping it. Only then does our conscious mind take control to decide what to do next. This approach has some biological justification [1].

Two sensing modules, vision and speech, handle all vision and speech-based interaction. Their main task is to act as filters between the sensory data and the symbolic information required by the rest of the system. Two reflex modules, navigation and face, handle the motion and appearance of the robot. Central control of the robot is handled through a state module, and communication between robots is handled through its own module. Finally, the architecture has two modules for debugging purposes. One, the monitor, shows text fields that represent all of the information available to the system. The other, the visual monitor, is designed to show graphically the information being provided by the vision module.

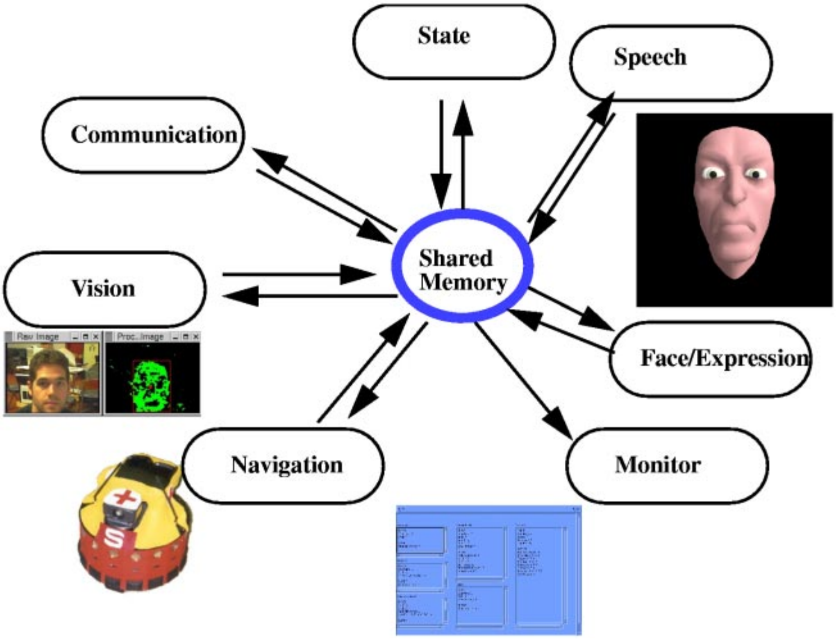

Fig. 1 Logical diagram of the REAPER Architecture. Each module takes inputs from and writes its outputs to the shared memory. The State module is the central controlling unit.

The modules in the architecture communicate through a shared memory structure, which provides a computationally efficient means of sharing information. They use a common framework for communicating and programming, including a handshaking protocol to ensure that information and commands are passed and read correctly. Communication between robots occurs through sockets between the agents' communication modules over a wireless ethernet system.

The REAPER design methodology follows a number of the lessons learned about robot architectures over the past two decades. In particular, it follows the recommendations of Bryson that an autonomous agent architecture needs:

- a modular structure,
- a means to control action and perception sequences for complex tasks, and
- a mechanism for reacting quickly to changes in the environment [4].

REAPER is related to behavior-based systems, such as those proposed by Brooks, in that it distributes low-level interpretation and control of sensors and actuators to modules [3]. REAPER is not a pure behavior-based system, however, because it is designed to facilitate the mixing of symbolic and subsymbolic, reactive strategies. Thus, REAPER is also related to hierarchical architectures, which facilitate the development of action and perception sequences for complex tasks.

The real strength of the system, however, is REAPER's modular design which facilitates cross-platform and multi-platform development and simplifies debugging and testing of the individual modules. This is particularly true in the case of the vision module, which we extensively tested on its own before putting it on the robot platform.

At the most fundamental level, the REAPER approach evolved because it enables the agent to gather information at a tremendous rate from a large number of sensors and present that information in a useful, symbolic form to the controlling module. This overall architecture for a perceptive agent is the framework within which we developed the real-time vision module. Our goals in designing the vision module were to avoid dominating the robot's processor, provide easy access to a wide variety of information about the world, and maintain as fast a frame rate as possible to avoid time lags between sensing and action. The remainder of the paper describes how we balanced these goals within an integrated robot system.

3 Vision Module

From the command module's point of view, the vision module is an information provider. The command modules specifies a general action mode and then specifies the operators from which it wants information. The vision module then begins processing images and continually updates information associated with each active operator.

The vision module itself begins with an initialization routine that sets up its shared memory and alerts the other modules it is active. It then enters an event loop--initially in an idle state. Each time through the event loop, the module tests if the controller has issued it a new command, either a new mode or a new set of operators to apply. If so, the transition to executing that command takes place. Otherwise, the module continues executing the current mode and operator set. The module continues to process and update sensory information until told to do something else. The goal of the vision module is to maintain 30Hz, or real-time visual processing, for any given operator set.

3.1 Modes and Operator Scheduling

The vision module includes a rich set of operators for converting images into symbolic information. The three general classes of operators are: object detection, motion detection, and object characteristic analysis. Each command to the vision module indicates a general mode and the set of operators that should be turned on. Other modules within the REAPER architecture can then scan the relevant output fields of the vision module for positive detections, motion, or object characteristics. Each output field includes information about where an object was detected in the image and when it was detected as determined by a time stamp. Based on the time stamp and the operator information, other modules can decide what information requires a response.

The set of operators provided by the vision module include:

- Person detection based on skin color and gradients
- Moving area detection across multiple frames
- Color blob detection, focused on colored conference badge detection
- P-similar pattern detection
- Red, white, and green flag detection
- Palm detection
- Orange arrow detection
- Green ring detection
- Shirt color analysis (dependent upon detecting a person)
- Person identification (dependent upon detecting a person)
- Calculation of how much food was on the robot's tray (using the tray camera)
- Take a panoramic image (on the Magellan robot only)

Which operators are available depends on the mode the controller selects. The major modes are: IDLE, LOOK, TRAY, and PANO. The LOOK mode is the primary mode of operation and permits all but the last two operators to be active. The TRAY mode activates the second camera input and analyzes how much of the robot's serving tray is filled. The PANO mode works with a pan-tilt-zoom camera to generate a 180° panoramic image that concatenates eight frames together while simultaneously applying the motion and person detection operators.

While in the LOOK mode, there is clearly no way to maintain a high frame rate and execute all of the operators on each image. To solve this problem the vision module uses a probabilistic scheduling algorithm that applies at most two operators to each frame. This solution works because the robot usually doesn't usually need to know that there is a pink blob in view--or whatever other object--30 times per second. Frame rate is usually a lot faster than the robot can react to things since reactions often involve physical actions or speaking. Likewise, most of the other operators do not benefit from continuous application in most human-robot interactions.

The scheduling algorithm is based on the premise that running two operators per frame will not reduce the frame rate. This puts an upper bound on operator complexity, although we can get around the limitation somewhat by pipelining the process. In the standard LOOK mode, the module randomly selects two of the active operators based on a programmer-defined probability distribution. To create the probability distribution, each process gets a weight associated with it, with processes requiring higher frame rates receiving higher weights. Most of the vision operators have small, relatively equal weights. Once selected, the module executes the two operators and updates their information. On average, each operator is executed according to the probability distribution defined by the manually supplied weights.

Since not all operators are applied on each frame, there can be a time lag between operator applications. To communicate the freshness of data to other modules, the vision module supplies a time stamp with each piece of information. The information persists until the next application of the operator, and the controller can decide based on the time stamp whether it is recent enough to warrant a response.

A secondary mode within the LOOK mode permits tracking--or consistent application of an operator--using one operator in addition to looking for other objects. To engage tracking, the controller specifies one tracking operator and the regular list of active operators. The vision module scheduler puts the tracking operator in one of the execution slots and randomly selects the other operator from the active list. This guarantees the vision module will look for the object being tracked every frame, providing the fastest update rate possible. In the tracking mode the navigation module can look directly at the vision module output and adjust its control of the robot accordingly. Mario, the Magellan Pro robot, could use this ability to follow pink badges.

The scheduling algorithm and overall structure were a successful way to manage our robot vision system. Even with all of the robot modules running, the vision module was able to maintain a frame rate of at least 20Hz during the competition. Information updates occurred often enough that the robots could attend to multiple aspects of their environment with real-time reactions.

To help us debug and monitor the vision module we developed a vision monitor module that watches the vision module output and graphically displays current information using color and bounding boxes. This enables us to determine how well the different operators are working.

A brief description of each operator follows. The new capabilities we developed for the 2000 competition included a new face detector, a short-term person identifier, a shirt-color identifier, a vertical Italian flag detector, and the ability to estimate how much food was left on the robot’s tray. For more complete descriptions of the other capabilities, see [11] and [12].

3.2 Motion-Based Person Detection

The motion detection operator is the most difficult operator to integrate within this framework because it requires multiple frames--at least three for robust processing--and requires a significant amount of processing for each frame. Our algorithm uses Sobel gradient operators to calculate edge images, and then subtracts adjacent (in time) edge images to locate edges that moved. It then locates the bounding box of areas of motion that exceed a certain threshold. We have found this algorithm to be quite successful at locating people in the hors d’oeuvres event [11][11][12].

To avoid breaking the overall structure of the vision module, we pipeline the algorithm across multiple event loops. The motion algorithm takes five event loops to complete--with the first three capturing images and calculating the Sobel results. To ensure the motion algorithm is called frequently enough, it has a high weight in the probability distribution. On average, the motion algorithm completes 5-6 times per second and the overall vision module maintains a frame rate of at least 20Hz. When the motion operator is active, it is usually selected as one of the operators.

3.3 Skin-Based Person Detection and Identification

We used two independent techniques, motion and face detection, to detect people. Our motion detector is described above, but we took a somewhat novel approach to face detection that resulted in a robust technique in the HA domain.

The basis of our face detection system is skin-color blob detection. The key to skin detection is effective training (or good color constancy, which is hard), since lighting conditions strongly affect colors. We developed a fast, interactive training program, shown in Figure 2(c), that gives the user direct feedback about how well the system is going to perform under existing conditions. The output of the training algorithm is an rg fuzzy histogram, where r and g are defined as in (1).

$$r = \frac{R}{R+G+B} \quad g = \frac{G}{R+G+B}. \tag{1}$$

A fuzzy histogram is a histogram with entries in the range [0, 1] that indicate membership in the colors of interest. To create a fuzzy histogram we generate a standard histogram and divide each individual bucket by the maximum bucket value [18].

We use fuzzy histograms to convert standard images into binary images that contain only pixels whose colors have high fuzzy membership values. For skin-color blob detection we train the fuzzy histogram on skin-color regions of several training images and then keep only pixels with membership values above a specified threshold. To get blobs we run a 2-pass segmentation algorithm on the binary image and keep only regions larger than a certain size [16].

The result of the blob detection is a set of regions that contain skin-color. In previous competitions we ran into trouble using just blob detection because the walls of the HA competition areas in 1998 and 1999 were flesh-tones. While this was not the case in 2000, there were other sources of skin-color besides people in the environment.

Our solution for distinguishing between background flesh-tones and true faces is to multiply the gradient magnitude of the image with the skin-color probability image prior to segmentation. The gradient magnitude image, however, is pre-filtered to remove high gradient values (i.e. strong edges). The result is a gradient image where mild gradients are non-zero and all other pixels are zero or close to it. Faces are not flat and contain mild gradients across most of their surface. However, they do not tend to contain strong edges. Thus, including the mid-range gradient values effectively eliminates walls--which are flat and tend to be featureless--but leaves faces. We found the combination to be robust and it reduced our false positive rate to near zero for the event while still reliably locating people.

3.4 Short-Term Person Identification

In the 1999 competition our robot, Alfred, tried to remember people based on texture and color histograms from a fixed portion of the image. Unfortunately, this relied on the person standing directly in front of the camera, which was rarely the case. This year we integrated the person identification with face detection and shirt color identification, described below. We did not store a permanent database of persons, but instead used a 100 entry buffer to recall people for a short time period. The purpose, therefore, of the person identification was to discover if a particular person was standing in front of the robot/agent for an extended period of time.

After a successful face detection the memory algorithm extracts a bounding box around the person’s body based on the location of their face, as shown in Figure 2(a). It then extracts a short feature vector from that box to represent that person’s identity. The feature vector is the top five buckets in an rg histogram--as defined in (1)--the top five buckets in an IB (Intensity, Blue) histogram, the average edge strength as determined by X and Y Sobel operators, the number of strong edge pixels, and the number of significant colors in the rg histogram. These 12 numbers provide a nice key with which we can compare people's appearance.

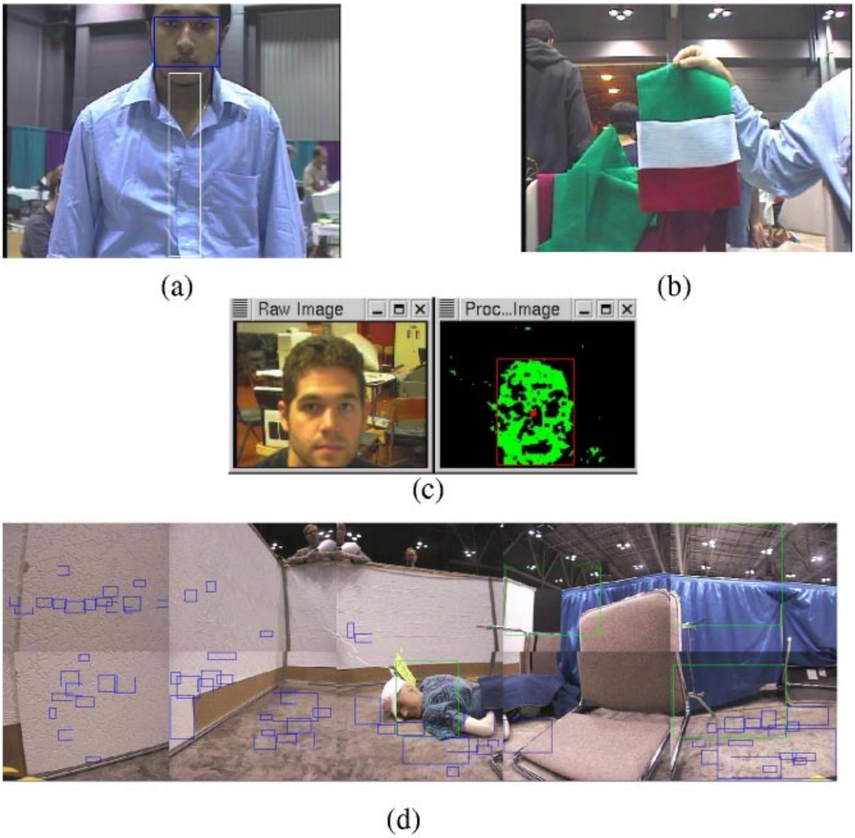

Fig. 2 Examples of the vision module in action. A) Successful face detection and the corresponding box used for shirt color and person identification. B) Successful flag detection. C) Training system for face detection system. D) Panoramic image from the USR contest: the green and blue boxes indicate possible motion and skin color respectively. Note that the skin-color on the mannequin's arm--on which we trained--was grey, which is why the walls and floor get highlighted.

To compare two keys, we use standard histogram intersection with the top five buckets in the rg and IB histograms, and then percentage difference in the average edge strength, number of strong edge pixels, and number of significant colors. Appropriate normalization provides a single number which indicates the degree of match.

Once the system extracts a key, it compares the key to all other keys recently seen. The system stores the 100 most recent unique keys. If it finds a probable match, then it will send the matching ID to an output filter. If it finds no match, it will add the key to the data base and then call the output filter with the new ID value. The output filter simply returns the most common key identified in the past 10 calls. However, if no single key has at least three matches in the past 10, it returns a null result (no match).

The output filter guarantees that, even in the presence of a moving person and schizophrenic face detection results (jumping between people), if a person is standing in front of the camera for an extended period of time their key will register consistently.

We used this capability with Alfredo, the maitre'd. If a person was standing in front of Alfredo for a minimum period of time, it would comment that they should go do something else. Clearly there are other applications, but we did not pursue them in the competition setting.

3.5 Shirt Color Identification

Like the memory operator, the shirt color operator waits for a successful person detection (face or motion) and then extracts a section of the image that corresponds to the likely location of the person's shirt, as shown in Figure 2(a). The algorithm then generates a histogram of this region and selects the dominant color. The difficult aspects of this task are selecting the histogram space, and attaching color labels to regions of that space.

Based on experimentation, we selected the rgI histogram space to represent color, where

$$I = \frac{1}{3}(R + G + B) \tag{2}$$

I is intensity, and r and g are normalized color as defined by (1). (R, G, B) are the raw pixels values returned by the camera. We used 20 buckets in each of r and g, and 4 buckets in I.

Because different camera settings and different lighting affect where a color sits in the rgI space, we calibrate the system using a MacBeth™ color chart prior to each situation in which the robot interacts--i.e. once prior to the competition. Using a picture of the color chart under the appropriate illumination we identify the centroid in the rgI space for each of the 24 colors on the chart.

After identifying the region of interest--the shirt region--the system identifies the most common color in the rgI histogram. The system then finds the closest color centroid in a Euclidean sense and returns its text color label as the output. Alfredo correctly identified numerous shirts during the competition, including Dr. Maxwell's mother, who was wearing a purple shirt. It made the computer appear cognizant of its surroundings in an engaging manner and provided it with something intelligent to talk about.

3.6 Food Tray Analysis

The food tray analysis is a simple, but effective algorithm. We have two cameras on the robot connected to an Osprey 100 framegrabber card which has multiple composite video inputs that are software selectable. Upon entering the TRAY mode, the vision module switches to analyzing the input from a small greyscale camera mounted on top of the food tray looking across it. During the competition we used a white napkin to cover the tray and served dark brown or black cookies.

The tray analysis algorithm works on the middle 1/2 of the image, in which the tray dominates the scene. The operator counts the number of dark pixels and calculates the percentage of the visible tray that is full. Using pre-calculated minimum and maximum values, the operator sets a flag that specifies FULL, EMPTY, or a percentage in between. This is a good proxy for how many cookies remain on the tray. Since the small camera includes an auto-gain feature, this method works even when someone blocks the direct lighting by leaning over the tray or standing so it is in shadow. This feature worked well and enabled the robot to respond to how much food was left on its tray by heading towards its food refill station in an appropriate manner.

3.7 Vertical Italian Flag (Red-White-Green) Detection

Finally, we gave the robots for the hors d'oeuvres event the ability to strike up conversations with one another. To make this capability realistic it should only happen when the robots can "see" one another, which means they must be able to visually recognize each other. Since our theme was an Italian restaurant, we used the italian flag colors--red, white, and green--as our identifying feature. Santino had a flag draped vertically from his serving tray, and Mario had one placed on an antenna about 4 feet above the ground. To differentiate the two mobile robots we reversed the order of the colors for Mario and Santino from top to bottom.

The technique we use for flag recognition is based on traversing columns in the image with a state machine that tracks the order and color of the pixels. The state machine only outputs a positive identification if it finds a vertical series of red, white, and green pixels (or in reversed order). Each color has to be mostly continuous and contain at least a specified number of pixels. The state machine allows a certain number of invalid (not red, white, or green) pixels as it traverses the colors. However, too many invalid pixels invalidates that particular recognition and resets the state machine. This method, since it is based on single columns, is robust and easy to execute in real time. The recognition system worked well both in test runs and in the competition.

4 Summary

The products of our experience that we will continue, and are continuing to use are the overall REAPER architecture, the navigation modules, the face module, and the vision module. All of these provide us with generic scaffolding on top of which we are building other capabilities and systems. All of them are extendable and easily integrated with one another. We also now have excellent debugging tools that permit us to track all of the information and messages that pass between modules during execution. For us, this infrastructure is the real outcome of this project.

In particular, we have continued to use and develop the vision module for research purposes. It is straightforward to add capabilities to the module and use it for a variety of tasks. The probabilistic scheduling algorithm and wide variety of capabilities make it ideal for a mobile robot that is processor limited but must continuously monitor and react to multiple aspects of its environment.

As shown by the motion operator, the vision module is not limited to simple operations. We can use more complex operators by pipelining the processing across multiple event loops. Ultimately we are limited by the processing power of the robot, but clearly operators of moderate complexity are feasible within the real-time vision module approach.

Finally, the tracking capability gives the module the ability to focus on one operator and provide fast updates while still actively monitoring the environment. The combination of fast image capture, the application of operators in a probabilistic manner, pipelining of complex operations, and the ability to track, avoids bogging down our robot's processor and permits it to respond appropriately to the visual world.

References

[1] E. Bizzi, S. Giszter, E. Loeb, F.A. Mussa-Ivaldi, and P. Saltiel, "Modular organization of motor behavior in the frog's spinal cord", *Trends in Neuroscience*, 18:442-446.

[2] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack, "Experiments with an architecture for intelligent, reactive agents", *J. of Experimental & Theoretical Artificial Intelligence*, 9(2/3):237-256, 1997.

[3] R. A. Brooks, "A robust layered control system for a mobile robot", *IEEE J. of Robotics and Automation*, vol. 2, no. 1, 1986.

[4] J. Bryson, "Cross-Paradigm Analysis of Autonomous Agent Architecture", *J. of Experimental and Theoretical Artificial Intelligence*, vol. 12, no. 2, pp 165-190, 2000.

[5] D. R. Forsey and R. H. Bartels, "Hierarchical B-spline refinement", in *Computer Graphics (SIGGRAPH '88)*, 22(4):205-212, August, 1988.

[6] E. Gat, *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*, Ph.D. thesis, Virginia Polytechnic Institute and State University, 1991.

[7] *IBM ViaVoice™ Outloud API Reference Version 5.0*, November 1999.

[8] E. C. Ifeachor and B. W. Jervis, *Digital Signal Processing. A Practical Approach*, Addison Wesley Publishing Company, 1995.

[9] D. Kortenkamp, R. P. Bonasso, and R. Murphy (ed.), *Artificial Intelligence and Mobile Robots*, AAAI Press/MIT Press, Cambridge, 1998.

[10] B. A. Maxwell, L. A. Meeden, N. Addo, P. Dickson, N. Fairfield, N. Johnson, E. Jones, S. Kim, P. Malla, M. Murphy, B. Rutter, E. Silk, "REAPER: A Reflexive Architecture for Perceptive Agents", *AI Magazine*, spring 2001.

[11] B. A. Maxwell, L. A. Meeden, N. Addo, L. Brown, P. Dickson, J. Ng, S. Olshfski, E. Silk, and J. Wales, "Alfred: The Robot Waiter Who Remembers You," in Proceedings of *AAAI Workshop on Robotics*, July, 1999. To appear in *J. Autonomous Robots*, 2001.

[12] B. Maxwell, S. Anderson, D. Gomez-Ibanez, E. Gordon, B. Reese, M. Lafary, T. Thompson, M. Trosen, and A. Tomson, "Using Vision to Guide an Hors d'Oeuvres Serving Robot", *IEEE Workshop on Perception for Mobile Agents*, June 1999.

[13] H. P. Moravec, A. E. Elfes, "High Resolution Maps from Wide Angle Sonar", *Proceedings of IEEE Int'l Conf. on Robotics and Automation*, March 1985, pp 116-21.

[14] J. Neider, T. Davis, and M. Woo, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, Addison-Wesley, Reading, MA, 1993.

[15] F. I. Parke and K. Waters, *Computer Facial Animation*, A. K. Peters, Wellesley, MA, 1996.

[16] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing", *ACM*, 13:471-494, October 1966.

[17] D. Scharstein and A. Briggs, "Fast Recognition of Self-Similar Landmarks", *IEEE Workshop on Perception for Mobile Agents*, June 1999.

[18] H. Wu, Q. Chen, and M. Yachida, "Face Detection From Color Images Using a Fuzzy Pattern Matching Method", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 6, June 1999.