REAPER: A Reflexive Architecture for Perceptive Agents

Bruce A. Maxwell, Lisa A. Meeden, Nii Saka Addo, Paul Dickson, Nathaniel Fairfield, Nikolas Johnson, Edward G. Jones, Suor Kim, Pukar Malla, Matthew Murphy, Brandon Rutter, Eli Silk

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

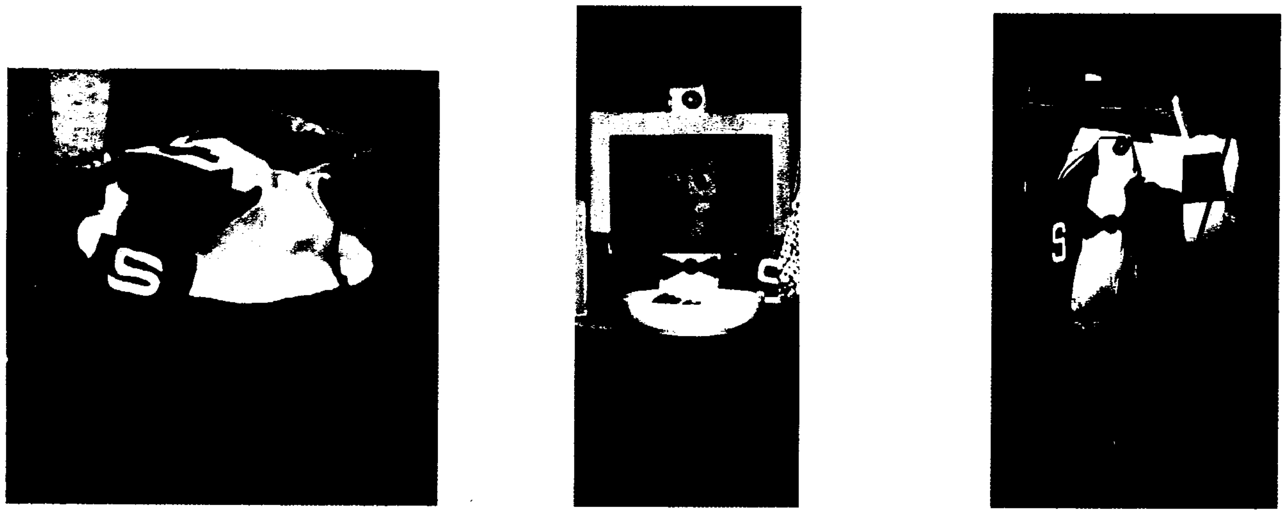

Figure 1 Alfredo (center) the maitre'd and his two "sons" Mario (left) and Santino (right). Mario is in his search and rescue uniform, while Santino is ready to serve hors d'oeuvres.

Abstract

For robots to interact naturally with people and their environment they need to have a variety of perceptual abilities. They need to be able to see, hear, feel and otherwise sense the world around them. The problem is not giving them the sensors, but rather giving them the ability to process and use the sensory information in real time. We have developed a platform independent modular sensory architecture that permits us to smoothly integrate multiple sensors, navigation, and communication systems. The architecture gives each module local control over a sensor or system. These modules act as filters between the sensor and a central controller that reacts only to higher level information and gives only high level commands. Using this architecture on multiple, communicating robots we were able to develop a unique trio of characters for the AAAI Hors d'Oeuvres Anyone event as well as an entry in the Urban Search and Rescue event.

1 Introduction

In 1999, Swarthmore's waiter robot, Alfred, won the American Association for Artificial Intelligence [AAAI] "Hors d'Oeuvres Anyone?" robot competition. This year, Alfred graduated to italian restaurant owner--changed his name to Alfredo--and went back to the competition with his "sons" Santino and Mario. Alfredo was the maitre'd, Santino the waiter, and Mario the bus-boy. They are shown in Figure 1.

This year Alfredo was not a mobile robot, but a computer with a large monitor placed at the waiter's refill station. He had speakers and a video camera, and would respond to different kinds of visual input. The monitor displayed a talking face, whose lips move in synchronization with the speech. He had three special capabilities: 1) he could tell when you held your palm in front of the camera and would give you a palm reading. 2) he would comment on the color of your shirt (based on analysis of the video image), and 3) he would comment if you stayed in front of the camera too long. Otherwise, Alfredo would talk about various things, responding to what he saw in the camera.

Santino, the waiter, was a Nomad Super Scout II, a medium size mobile robot with an on-board computer. Santino was also outfitted with two cameras, a microphone, speakers, a 6" LCD display and a mechanical arm that could raise a tray up and down. Santino used the two cameras to look for people, look for brightly colored badges, and to check when his tray was empty. He would come up to a person, ask if they wanted an hors d'oeuvre and then lift the tray if they said yes. When his tray was empty, he would make his way back to the refill station. When Santino was happy a face on the LCD screen would smile. When he was grumpy or angry, it would frown.

Mario, the bus-boy, was a Real World Interfaces [RWI] Magellan Pro, a short mobile robot with a camera and speakers. His job was to provide entertainment by running around in the crowd. During the competition, he also had a plate of cookies on his back. In addition, he would shuttle back and forth between Santino and Alfredo, attempting to strike up conversations with them. The two mobile robots could identify one another by a red, white, and green flag that each carried (one with the red side up, one with the red side down).

This year Swarthmore not only competed in the "Hors d'Oeuvres Anyone?" event, but also in the Urban Search and Rescue [USR] event on a standard course prepared by the National Institute of Standards and Technology [NIST].

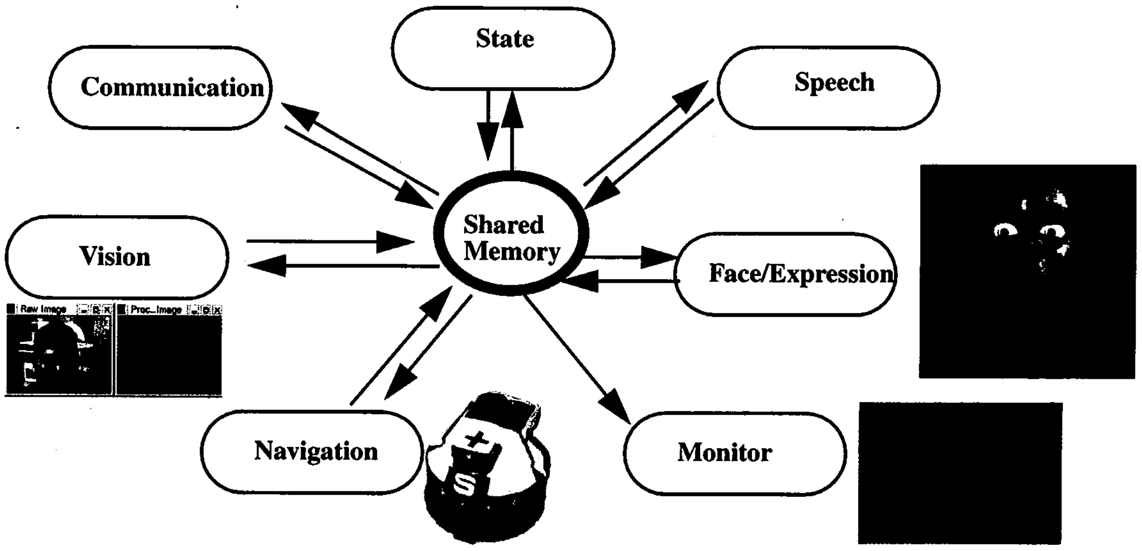

Figure 2 Logical diagram of the REAPER Architecture. Each module takes inputs from and writes its outputs to the shared memory. The State module is the central controlling unit.

The robot Mario explored one section of the course autonomously, built a map, and connected annotated 360° panoramic images of the scene to map locations. The annotations identified image areas of interest by highlighting motion and skin-color. Mario then made its way out of the course within the allotted time limit (25 minutes).

Even with ten undergraduate students working on the project for eight weeks, doing both events at this level of performance was difficult. What made it possible, let alone successful, was that each of the agents used the same overall software architecture for integrating navigation and control with perceptual processing. Furthermore, this architecture was designed to be largely platform independent and modular, permitting different agents--including non-mobile agents--to use different capabilities with few changes to the overall system.

Using the same architecture for each agent allowed us to distribute our efforts and focus on common capabilities such as visual information processing modules and facial animation modules that could be used on several platforms. This permitted us to give each agent a wide range of abilities and then integrate them together effectively. The unique aspects of our hors d'oeuvres entry this year included:

- The integration of multiple sensors and modes of interaction in a single agent,
- A powerful, general purpose, real-time color vision module,
- Fast, creative, entertaining, and robust human-agent interactions,
- Facial animation--including tracking faces with the eyes--in sync with the text,
- Shirt color detection and identification,
- Fast, safe navigation in a crowded space using a reactive algorithm, and
- Communication and interaction between agents.

The same architecture also managed our USR entry. The only difference between Mario the bus-boy and Mario the rescue robot were the controlling modules. Otherwise, the vision, speech, and navigation modules were identical. The strengths of our USR entry were:

- Completely autonomous function,
- A robust reactive wander mode and "get out" mode,
- Building a map of the environment with connected annotated images, and
- The effective use of the same overall architecture.

The rest of this paper examines the overall architecture and highlights the most important pieces.

2 REAPER: an Intelligent Agent Architecture

The 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 of: 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 ten times per second. At the same time it does need 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 intermediate actions between high-level commands or goals. This is similar to the way our brain deals 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 back control to decide what to do next. A graphical representation of the architecture is shown in Figure 2.

The two sensing modules 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. The reflex modules--navigation and face--handle the motion and appearance of the robot. The navigation module also incorporates sensing (sonar and infrared sensors), but its primary task is to control the motion of the robot, not to filter the sensory information. Central control of the robot is handled through a state module, and communication between robots is handled through its own module. Finally, we created 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 graphically show the information being provided by the vision module.

The modules on a robot communicate through a shared memory structure, which provides an efficient means of sharing information. They are based on a common framework for communicating and programming that uses a handshaking protocol to ensure that information and commands are passed and read correctly. Communication between robots occurs through sockets between the communication modules over a wireless ethernet system.

Central control of the robot was handled by a controller module, or state module. This module was started first, and it would start up all of the other modules it needed--each which of which was its own program. The state module would then initiate a state machine process that specified how the robot would interact with the world, what sensing and interaction modalities it would use, and what kinds of navigation it needed to accomplish. To specify what the other modules should do it used a handshaking protocol to send information and commands to them. The other modules, in turn, would maintain blocks of output information that could be used by the state machine to determine what to do next and when certain actions were complete.

The state machine design and implementation required careful planning and thinking. The most difficult aspect of developing them was synchronization and timing. The state machine used a handshake protocol involving two counters--one controlled by the state machine, one by the module--to synchronize commands with a given module and ensure it didn't send commands too quickly. The state machine also had to be carefully constructed so that it didn't switch between states too quickly. Since the state machine did not include any of the low-level sensing or interaction, it iterated extremely quickly and could move between states before other modules had any chance to react to the previous state. Thus, it had to watch flags from the other modules to determine when actions completed before moving on or making a decision. The strength of this approach is that the state machine can sit back and sample high-level information asynchronously, reacting to changes in the world smoothly and quickly.

2.1 Overall module structure

The non-controller modules all contained the same basic program structure. After startup and initialization, each would enter an event loop--initially in an idle state. Each time through the event loop, the module would first test if the controller had issued a command. If so, the transition to executing that command would take place. Otherwise, the module would process the current command. When it completed the current command, the module would transition itself back to an idle state and indicate to the controller via a flag that it was in an idle state. In some cases, such as sensing commands, the module would continue to process and update sensory information until told to do something else.

The goal of all of the modules was to make the event loop as fast as possible. In the navigation module, the goal was to maintain a control loop of at least 10Hz; in the vision module, the goal was to maintain 30Hz, or real-time visual processing.

2.2 Reflexes: Navigation

The navigation modules on the Scout and Magellan had to be platform-specific because of the differences between the two robot's low level interfaces. From the point of view of the controller modules, however, they appeared similar. Different groups developed the navigation modules, so, while they are both reactive, they differ in their specifics.

2.2.1 Scout Navigation The navigation requirements for the scout were simple. It had to move slowly and safely, be able to get to a goal location, and be able to avoid obstacles. In addition, it had to have a mode where it actually stopped for an obstacle in case it was a person to serve.

The navigation module was setup as a 2-layer reactive system. The sensors available to the navigation module were the sonars and bump sensors, including five bump sensors on a low front bumper we added to Santino. The bottom layer contained a set of behaviors that reacted directly to these inputs. These behaviors included the following.

- Goal achieving
- Obstacle avoidance
- Wander
- Free-space finding
- Front bumper reaction

Each of these behaviors would return a fuzzy priority, speed, and heading. The controller layer would then combine the speed and heading values based on its mode and the currently active behaviors.

The modes/commands for the navigation system included: Idle, Stop now, Stop slowly, Goto avoid, Goto attend (stop for obstacles), Put the arm up, Put the arm down, Wander, Track attend, Track avoid, and a set of commands for setting the odometry and controlling orientation.

The most interesting of these modes were the track modes. The intention here was to create a mode that would directly connect the vision system and the navigation system without controller intervention. It could be used to follow a judge's name-tag badge or track a target in real-time. Once the vision module found a badge or target, the controller could initiate the mode in both the vision and navigation modules. Once initiated, the vision module would continue to track the object and update the object's position. The navigation module, in turn, would react as quickly as possible to the visual information and try to orient and follow the target. It would continue to track the target until either the target was lost, the controller ended the tracking, or an obstacle appeared (in the case of Track Attend).

2.2.2 Magellan Navigation The Magellan Pro--Mario--is a small round robot with symmetrically opposed wheels which allow it to rotate on its axis. The basic sensor array consists of a three rings of 16 bump (contact), sonar, and IR sensors mounted around the sides of the robot. In addition Mario has a Sony DV30 pan-tilt camera and external speakers. The on-board computer is a Pentium II running Linux 2.2.10, and communicates with the robot's rFlex controller over a 9600 baud serial line.

Because of the lack of a low-level software library, we developed an interface for the Magellan which we called Mage. Mage communicates directly with the rFlex controller of the robot. The rFlex accepts a simple set of motor control commands and is also responsible for transmitting the sensor data of the robot back over the serial line. We were able to extract or deduce most of the protocol for this communication from some example code that RWI provides for updating the CMOS on the rFlex. At our request, RWI sent us code snippets containing information relevant to the IR sensors, which allowed us to enable and read the IR range values. During this time we also developed and integrated a controller for the Sony pan-tilt-zoom camera, which was controlled over a separate serial line.

In general the Mage API closely resembles the API for the Nomad SuperScout (due to the fact that we have extensive experience with the scouts), although we implemented a simplified command set and decided to make the units of distance thousandths of meters and the units of rotation thousandths of radians.

In keeping with the Nomad API, all sensor and motor control data is maintained in a large state vector. For example, the statement State[STATE_SONAR_0] returns the most recent value of the forward-pointing sonar sensor. This state vector is updated continuously by a thread which handles new data passed from the robot controller. Although the rFlex controller supports a request-based protocol, the simpler method is to ask it to continuously stream data from the sensors as fast as it can. This approach ensures that the sensor data is as up to date as possible. In order to send motor commands, the API includes a method which sets the contents of an output buffer. The same thread which handles incoming data also watches this buffer and transmits its contents to the rFlex controller. As a note, this motor data is transmitted immediately if it changes and then transmitted periodically to keep the rFlex controller alive. The serial communications to the pan-tilt-zoom mount of the camera is implemented in the same way.

The navigation module sits on top of the Mage API and is responsible for reporting the basic sensor data and for actually getting the robot from point A to point B without running into anything. In our implementation, the nav module had several different modes, but they were all based on a reactive kernel. The robot decided how much to translate and rotate based on four lines of code.

- Translate = Translate - Distance to closest front object
- Translate = Translate + Distance to closest rear object
- Rotate = Rotate - Distance to nearest object to the right (assuming clockwise rotation)
- Rotate = Rotate + Distance to nearest object to the left

To make the robot wander, we just had to give Translate a forward bias. To go to a goal point, we calculated the Translation, Rotation bias required to push the robot towards the goal point. To track an object, the navigation module monitored the relative position of the object (stored in the vision module), and fed this information straight into the biases. This approach proved to be very robust as long as the biases did not exceed the maximum repulsion of obstacles.

To build a map in the USR event, the navigation module used an evidence grid approach (Moravec and Elfes, 1985). We integrated sonar readings into a probabilistic map that could then be classified into free space and obstacles for interpretation by a person. The evidence grid technique worked well in our test runs, but in the actual event small objects on the floor and tight paths between obstacles caused sufficient wheel slip to significantly throw off the odometry. Thus, local areas of the map were correct, but globally it did not reflect the test situation.

2.3 Reflexes: Face

Robot-Human interaction is the key component that distinguishes the Hors d'Oeuvres Anyone? competition from other robot competitions. the goal of creating a fully-functional intelligent agent with the capabilities of any average human is far from realized. Yet our robot team this year began to make strides in developing our own synthetic character to better solve the difficult task of the competition by incorporating an animated, 3-D graphical model of a human head with interactive capabilities.

A growing amount of work has been dedicated to the creation of synthetic characters with interesting interactive abilities. Each year the competitors in the robot contest find better ways to explicitly display complex interaction s with humans. We considered a number of graphical models with the capability to display emotion and the flexibility to add increasingly more complex abilities. The Dragon Wing, for example, is a facial modeling and animation system that uses hierarchical b-splines for the generation of complex surfaces (Forsey and Bartels, 1988). The technique provides an incredible amount of flexibility, but was too complicated for our needs. Instead we utilized a muscle model for facial animation and facial geometry data available on the web (Parke and Waters, 1996). We ported the system to OpenGL on Linux (Neider, Davis, and Woo, 1993).

The facial model is a simple polygon representation that uses 876 polygons. Only half the face is actually described in the input data file since symmetry is assumed between the right and left sides. Reading the data and rendering it is straightforward in OpenGL. The system we developed permitted the user to view the face data in a number of ways, including: transparent, wire frame, flat shading, and smooth shading. In addition, the face could be oriented and rotated by the user.

The model we used included a simple muscle model to animate the face. A second data file defines the muscles by specifying the beginning and ending points, as well as a zone of influence. Each muscle can be relaxed or contracted, affecting all those vertices within its specific zone of influence. We created a set of predefined expressions which consisted of a set of contractions for each muscle in the facial structure. We could move between expressions by interpolating the differences in the expression vectors. Our system used a total of 18 different muscles and 6 unique expressions. Each of the expressions is shown in Figure 3.

Beyond the structure of the face, we added a couple of features to increase the interactivity of the system. First, we gave the jaw the ability to move in order to synchronize

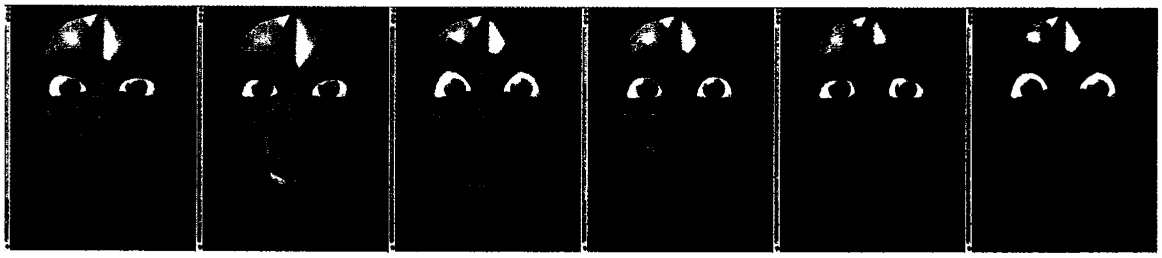

Figure 3 The faces of Santino. From left to right: anger, disgust, fear, happy, sad, surprised.

mouth movement along with speech generation. The jaw was able to rotate vertically by specifying jaw polygons and then rotating them about a central axis. The mouth was also able to move horizontally from puckered lips to a wide mouth by adding a virtual muscle that contracted the polygons of the mouth. Our speech generation program, IBM's ViaVoice™ Outloud, generated a mouth data structure--containing mouth height and width--in parallel to the sound synthesis. We passed this information to the face module and used it to update the mouth state in synchronization with the robot's speech.

The second capability we added was to give the face eyes--half-spheres colored appropriately with an iris and pupil. We then transformed the eyes according to the output of the vision module. This simulated the effect of the eyes tracking people's faces or focusing on their conference badges.

We presented the faces on Santino and Mario using color LCD displays at a resolution of 640x480 in 8-bit color. On Alfredo--a dual processor workstation--we presented the face on a 17" monitor with 8-bit color at a resolution of 800x600 pixels. The complete animation capabilities were only used on Alfredo because of the more limited processing power on the mobile robots. On Alfredo, with the full capabilities--and the vision module running simultaneously--the rendering system was able to run at approximately 9 Hz, which was at the low end of acceptable quality.

Overall, the facial animation system greatly enhanced the interactively capability of the trio of intelligent agents. It gave people a central focus when interacting with the robots and helped to keep their interest throughout the interaction.

2.4 Senses: Speech

To serve people, a server must be capable of interacting with those being served. This interaction can take several forms, but somehow communication must take place. The server must signal his/her presence and offer the objects being served, the servee must be able to signal acceptance, and the server must serve. On Santino, we chose to make the main modality of communication speech. To create a full interaction, we wanted Santino to be capable of asking people if they wanted an hors d'oeuvre, and responding correctly to their response. This required that we use both speech generation and recognition. We elected to use commercially available development software to accomplish both of these goals. For recognition, we elected to largely build on the development done for Alfred at the 1999 competition, development based on ViaVoice™ SDK for Linux. For speech synthesis, we decided that ViaVoice™ Outloud enabled us to do all the things we wished to do in addition to being easy to integrate with the ViaVoice™ recognition system.

There were several major problems to be overcome in developing the complete speech module. We decided that doing speech recognition in the actual competition was extremely important, though very difficult. ViaVoice™ software is designed for highly specific circumstances: a single person speaking clearly into a microphone in a mostly quiet room. The hors d'oeuvres competition was certainly not that. Instead, we could expect several hundred people chatting amongst themselves, and some people not knowing to speak directly into the microphone. Therefore, we needed to keep recognition interactions extremely brief and do whatever we could to get a clear sound signal for recognition.

Given that recognition even on monosyllables was going to be difficult, we wanted to make sure that the robot could be an interesting conversationalist. We wanted to avoid a stereotypical robotic voice, yet enable dialogue to be easily written and added. Additionally, it was important to us that the voice be able to express different emotions, especially as we planned to closely link speech with the expressive face module. Fortunately, Outloud enabled us to implement all these synthesis features.

Finally, we needed to make generation and recognition work on the actual mobile robot, with little processing power, system noise, and a poor sound card. Making ViaVoice™ and Outloud work together with poor audio processing equipment turned out to require extra levels of care.

2.4.1 Santino's speech module Our approach to recognition was much the same this year as in 1999 (Maxwell et al., 1999). Though ViaVoice™ can be made to recognize complex grammars with large vocabularies, it has difficulty with recognition in noisy environments. Therefore, doing anything approaching complex speech recognition was not reasonable under the competition circumstances. We decided therefore that the robot primarily needed to understand simple yes-no type responses, and simple polite words, like please or thanks. Therefore we tailored our efforts in recognition towards getting high recognition rates on these monosyllables, rather than attempt to hold a more complex conversation.

One of the major improvements on the speech system that was suggested by last year's hors d'oeuvres competition was to allow our robotic waiter agent be able to detect when the background noise exceeded a threshold and made it undesirable for speech recognition. With this added ability, we could program our robotic waiter to simply shut down its speech recognition component and switch into a different mode that only used speech synthesis. This noise detection ability would greatly improve speech recognition rates since the robot would attempt recognition only in reasonable environments.

We were able to implement this background noise detection feature through a simple signal processing technique (Ifeachor and Jervis, 1995). We implemented a routine that calculated the average power of a ten second sound recording from an omni-directional microphone and compared it to threshold values. These threshold values were determined at the conference hall some minutes before the competition. In determining appropriate threshold values, the peak power of a sound waveform was used as a guide to prevent us from specifying a threshold that would never be exceeded. Our threshold value was such that speech recognition could still occur with some amount of background noise.

In addition to making our speech module more robust, a simple Finite Impulse Response band-pass filter was implemented to eliminate frequencies that were beyond a specified range (~200Hz - 2kHz) (Ifeachor and Jervis, 1995). Mechanical objects--like ventilation fans in a conference hall--mainly produce the low frequencies, while high frequencies occur from electrical interference in the sound card--which is integrated on a single board computer. To ensure module independence and speed, we modified the ViaVoice™ Speech Recognition audio library to include the band--pass filtration. This bypassed the necessity to first record the speech utterance to a pulse code modulated (PCM) wave file, perform filtration and then pass the output to the recognition engine.

The most important part of the competition for Santino was interacting with a person during a serving scenario. As doing complex speech recognition was not a possibility, we devoted most of our energy to developing the robots spoken personality. We attempted to make the robot sound emotional, and to say properly emotional things. Originally, we planned to make emotion a very important part of speech, and have the robot enter each interaction with an emotional state, perhaps even having that emotional state change as a result of the interaction. In the end, we did not have enough time to tie emotions to causes within the environment, though that will certainly be a future goal. The robot still sounded emotional, and said emotionally charged things, but the emotional state was randomly determined.

There were several classes of spoken phrases used during the each serving scenario. When the state machine signaled speech to begin an interaction, it would say something that asked the person if they would like something to eat, often in an interesting and occasionally rude way. When the robot finished speaking, the recognition engine would be given control of the sound device, to record the response of the person. If a yes or no response was registered, the speech module would report the response to state, who would then instruct speech to respond appropriately and end the interaction. If there was a failed recognition, the robot would either say something about the color of the persons shirt--if vision had managed to detect shirt color--or something noncommittal. Santino would then ask the person again if they wanted an hors d'oeuvre and listen for a response. A second failure would cause speech to say something to just get out of the interaction, and state would look for someone else to serve. If the robot heard nothing at all, the speech module would comment that the person was probably a box being mistakenly served and move on.

When Santino was not in an interaction, he muttered, which was a running commentary about whatever the robot was doing at that moment. When the robot was in the GOTO_SERVE state and not serving anyone, it would mutter about all the food that it had to give. In the GOTO_REFILL state, it would mutter and tell people to not bother it; there was no food to be had. We had to overcome several problems to get this to function properly on the actual robot. In particular, we had to make synchronous calls to both ViaVoice™ programs telling them to stop controlling the audio device in order to deal with a slow turnaround time switching from input to output on the sound card.

The speech module acquitted itself very well at the competition. Recognition rates in the crowded hall were fairly high, at about 70-75%, which included misrecognitions of people not talking into the microphone, or saying something with absolutely no resemblance to yes-no responses. Given the loudness and the large numbers of people, the robot did just a little worse than a human might have in the same circumstance. The worst mistakes were made when it appeared that a variable was not getting properly cleared, causing the robot to respond to a no response as if it were a yes response, but this only seemed to happen once or twice. Most problems had been isolated during extensive testing of speech apart from the other modules, where it performed almost perfectly.

2.4.2 Mario's speech module Because Mario did not attempt speech recognition, its speech module was a simplified version of Santino's. The speech module mainly served a diagnostic function, encoding information about the internal state of the robot into natural-sounding phrases, as well as a means for the robot to communicate its goals and interact with humans. The speech output is expressed as strings and then we render the speech using IBM's ViaVoice™ Outloud. Although the speech module does have the functionality to read and speak a phrase directly from the state module, we often used a more flexible mutter mode. In the mutter mode the speech module monitors the shared memory information fields and makes its own decisions about what to say. Once properly configured, the mutter mode picks an appropriate phrase out of a pool of possibilities every few seconds. To a practiced ear this is informative about the robot's internal state but at the same time it reduces the risk of hearing the same boring phrase over and over.

2.5 Senses: Vision

Being able to sense the visual world gives numerous advantages to a robot, especially one involved in human interaction. Visual capability allows the robot to find and locate objects, detect motion, and identify visual object characteristics. One of our goals in both contests was to make the robots react to their world as quickly as possible. Thus, the navigation module maximized the number of times per second it executed the control loop. Likewise, our goal in the vision module was to maximize the frame rate while still providing a rich array of information.

The structure of the vision module was similar to the others. After initialization, the event loop checked if there was a pending command from the controller. If there was, it would transition to the new state according to the command. Otherwise, it would continue to execute the current command set.

The vision module included a rich set of operators for converting images into symbolic information. The three general classes of operators were: object detection, motion detection, and object characteristic analysis. Each command to the vision module indicated a general mode and the set of operators that should be turned on. The controller could then scan the relevant output fields of the vision module for positive detections, motion, or object characteristics. Each output field included information about where an object was detected in the image and when it was detected as determined by a time stamp. The controller could then decide what information required a response.

The set of operators we implemented included:

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

Which operators were available depended on the mode the controller selected. The modes relevant to the competition were: IDLE, LOOK, TRAY, and PANO. The LOOK mode was the primary mode of operation and permitted all but the last two operators to be active. The TRAY mode activated the second camera input and analyzed how much of the tray was filled. The PANO mode worked with the pan-tilt-zoom camera on Mario to generate a 180° panoramic image that concatenated eight frames together while simultaneously applying the motion and person detection operators.

While in the LOOK mode, there was clearly no way we could maintain a high frame rate and execute all of these operators on each image. Our solution was to devise a scheduling algorithm that only applied a few operators to each frame. This came about because of the realization that the controller didn't really need to know that there was a badge in view--or whatever other object--30 times per second. That was a lot faster than the robot could react to things since reactions generally involved physical actions or speaking. Running the badge detection 2-6 times per second was probably still overkill. Likewise, most of the other operators did not benefit from continuous application. Since we supplied a time stamp with each piece of information, the controller could decide based on the time stamp whether a piece of information was recent enough to warrant a response.

Our scheduling algorithm was based on the premise running two operators per frame would not reduce the frame rate. This put an upper bound on operator complexity, although in the case of motion analysis we got around the limitation by pipelining the process. In the standard LOOK mode, the module would randomly select two of the active operators based on a probability distribution. To create the probability distribution, each process was weighted, with processes requiring higher frame rates receiving higher weights. Most of the operators received small, relatively equal weights. Once selected, the module would execute the two operators and update the relevant information. On average, each operator would be executed according to the probability distribution.

The motion detection operator was the most difficult operator to develop 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 used Sobel gradient operators to calculate edge images, and then subtracted adjacent (in time) edge images to locate edges that moved. It then located the bounding box of areas of motion that exceeded a certain threshold. We have found this algorithm to be quite successful at locating people in the hors d'oeuvres event (Maxwell et al., June 1999)(Maxwell et al., July 1999).

We didn't want to break the overall structure, so we pipelined the algorithm across multiple event loops. The motion algorithm took five event loops to calculate a result--with the first three capturing images and calculating the Sobel results. To ensure the motion algorithm was called frequently enough, we gave it a high weight in the probability distribution. On average, the motion algorithm produced a result 5-6 times per second. When it was active, it was usually selected as one of the two scheduled operators.

A secondary mode with the LOOK mode permitted tracking using one operator in addition to looking for other objects. To engage tracking, the controller would specify a single tracking operator and the regular list of other active operators. The operator scheduler would then put the tracking operator in one of the two execution slots and randomly select the other operator from the active list. This guaranteed that the vision module would look for the object being tracked every frame, providing the fastest update rate possible. As noted above, in the tracking mode the navigation module could look directly at the vision module output and adjust its control of the robot accordingly. Mario used this ability to follow badges during the competition.

The scheduling algorithm and overall structure were extremely successful as a way to manage a robot vision system. Even with all of the other robot modules running, the vision module was able to maintain a frame rate of at least 20Hz. Information updates occurred regularly enough that the robot was able to attend to multiple aspects of its environment with real time reactions.

The new capabilities and algorithms we developed this year were: person detection and identification, shirt color identification, food tray analysis, and Italian flag detection. For details on the motion, color blob, and P-similar pattern detection see (Maxwell et al., June 1999), (Maxwell et al., July 1999), and (Scharstein and Briggs, 1999). Examples of each of the new capabilities are shown in Figure 4.

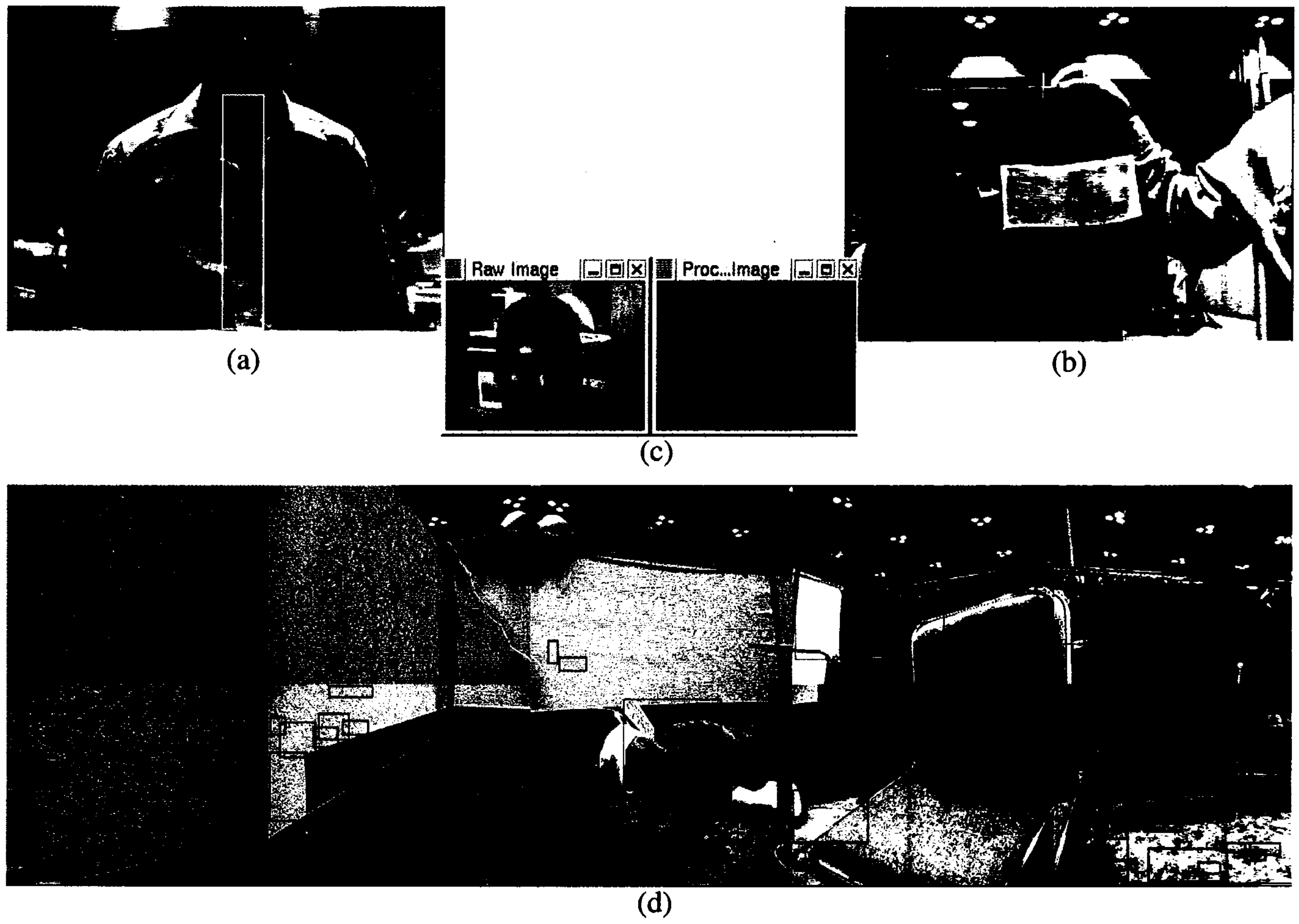

Figure 4 Examples of the vision module in action. A) Successful face detection and the corresponding box used for shirt color and person identification. B) Successful flag detection. C) Training system for face detection system. D) Panoramic image from the USR contest: the green and blue boxes indicate possible motion and skin color respectively. Note that the skin-color on the manniquin's arm--on which we trained--is grey, which is why the walls and floor get highlighted.

2.5.1 Person detection and identification Person detection is one of the most important capabilities for an interactive robot to possess. We used two independent techniques to accomplish this: motion and face detection. Our motion detector was straightforward, but we took a slightly novel approach to face detection that resulted in a fairly robust technique in the hors d'oeuvres domain.

The basis of our face detection system is skin-color blob detection. The key to skin detection is effective training, since lighting conditions can strongly affect the appearance of colors. We developed a fast, interactive training algorithm 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).

$$r = \frac{R}{R+G+B} \quad g = \frac{G}{R+G+B}. \qquad (1)$$

A fuzzy histogram is a histogram with entries in the range [0, 1] that indicate membership in the colors of interest. You can create a fuzzy histogram by taking a standard histogram--which counts the occurrences of each rg pair--and dividing each bucket by the maximum bucket value in the histogram (Wu, Chen, and Yachida, 1999).

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 some 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.

The result of 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 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 to this problem was to multiply a gradient image with the skin-color probability image prior to segmentation. The gradient image, however, was pre-filtered to remove high gradient values (i.e. strong edges). The result was a gradient image where mild gradients were non-zero and all other pixels were 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 gradient values effectively eliminated walls--which are flat and tend to be featureless--but left faces. We found the combination to be robust and it reduced our false positive rate to near zero while still locating people.

In the 1999 competition our robot--Alfred--tried to remember people based on texture and color histograms. This worked ok at the competition, but it relied on the person standing directly in front of the camera, which was rarely the case. This year we decided to integrate the person identification with the face detection and shirt color identification. We also decided not to store a permanent database of persons, but instead to only 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, if the memory feature was activated and called then the memory algorithm extracted a bounding box around the person's body based on the location of their face. It then extracted a short feature vector from that box to represent that person's identity. The feature vector was 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.

Once the system extracted a key, it compared the key to all other keys recently seen. The system stored the 100 most recent unique keys. If it found a probable match, then it would send this to an output filter. If it found no match, it would add the key to the data base and then call the output filter. The output filter simply returned the most common key identified in the past 10 calls. If no single key had at least three matches in the past 10, a null result (no match) was returned. The output filter guaranteed that, even in the presence of a person's motion and schizophrenic face detection results (jumping between people), if a person was standing in front of the camera for an extended period of time their key would register consistently.

We ended up using this information with Alfredo. If a person was standing in front of Alfredo for a minimum period of time, he would comment that they should go do something else. Clearly there are other applications, but we could not pursue them for lack of time.

2.5.2 Shirt color identification The shirt color recognition depended upon a successful face (skin) detection. Once a face was detected, the algorithm selected a section of the image below the face that corresponded to the person's shirt. The algorithm then analyzed a histogram of this region to determine the dominant color. The difficult aspects of this task were selecting a histogram space to use, and attaching color labels to regions of that space.

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

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

I is intensity, and r and g are the normalized coordinates defined by (1).

(R, G, B) are the raw pixels values returned by the camera for a given pixel. The benefit of using the rgI space is that the color--represented as rg--is then independent of the intensity--represented in the I axis. 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 calibrated the system using a MacBeth™ color chart prior to each situation in which the robot would interact. Using a picture of the color chart under the appropriate illumination we identified the centroid in the rgI space for each of the 24 colors on the color chart.

After identifying the region of interest--i.e. the shirt region--the system identified the most common color in the rgI histogram. The system then found the closest--in a Euclidean sense--color centroid and returned its text color label as the output. Alfredo used this system to great effect during the competition. It correctly identified numerous shirts, including Dr. Maxwell's mother, who was wearing a purple shirt. It made the computer appear cognizant of its surroundings in an engaging manner.

2.5.3 Food tray analysis The food tray analysis was a simple, but effective algorithm. We used an Osprey 100 framegrabber card with multiple composite video inputs. Upon entering the TRAY mode, the vision module would switch to analyzing the input from a small greyscale camera mounted on the tray. We used a white napkin to cover the tray and served dark brown or black cookies.

The tray analysis algorithm worked on the middle 1/2 of the image, in which the tray dominated the scene. Then we simply counted the number of dark pixels and calculated the percentage of the visible tray that was full. By having pre-calculated minimum and maximum value, we could control a flag that specified FULL, EMPTY, or a percentage in between. This turned out to be a good proxy for how many cookies remained. Since the small camera included an auto-gain feature, this method worked even when someone blocked the direct lighting by leaning over the tray or standing so it was in shadow.

Based on the percentage full values returned by the vision module, the controller was able to smoothly transition from pure serving, to serving while heading towards the refill station, to heading directly to the refill station because the tray was empty.

2.5.4 Vertical Italian flag (red-white-green) detection

One of the capabilities we gave the robots for the hors d'oeuvres event was the ability to strike up conversations with one another. To make this realistic it should only happen when the robots are close to one another. To ensure this we decided to give the robots the ability to recognize one another. We originally considered putting p-similar patterns--easily recognizable targets--on each robot. However, this would have detracted from the robot's appearance, which was something close to formal dress.

Since our theme was an Italian restaurant, we decided to use 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. Alfredo could also initiate conversations when he saw one of the mobile robots in his camera. To differentiate the two we reversed the order of the colors for Mario and Santino from top to bottom.

The technique we used for recognition was based on traversing columns--since the colors were arranged vertically. Along each column a state machine tracked the order of the pixels. The state machine would only output a positive identification if it found a vertical series of red, white, and green pixels (or in reversed order). Each color had to be mostly continuous and contain a sufficient number of pixels. The state machine allowed a certain number of invalid (not red, white, or green) pixels as it traversed the colors. However, too many invalid pixels invalidated that particular state traversal.

This method, since it was based on single columns, turned out to be extremely robust and could execute in real time. The recognition system worked well both in test runs and in the competition. Because Santino was almost continuously engaged in serving during the competition, however, it was never able to respond to Mario. For us, watching the robots engage one another prior to the competition was one of the highlights of the experience.

3 Lessons learned and looking to the future

The products of our experience that we will continue--and are continuing--to use are the overall architecture, the navigation modules, the face module, and the vision module. All of these provided 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.

What we also learned is that designing the controller module is still more art than science. From a practical point of view, if we continue to use the state machine approach we need to build a set of standard techniques for managing and passing information around the system. Some of this we have started, but it needs to be approached in a more formal manner. One alternative is to develop a generic state controller that uses a knowledge management system and a set of rules to determine its actions. This method would implement a three-layer architecture where the controller sits between a reactive system and a deliberative symbolic system (Kortenkamp, Bonasso, and Murphy, 1998).

Looking to the future, if the Hors d'Oeuvres Anyone? event continues then the challenge is to push the envelope. On the interaction front, one challenge is to develop a more generic speech interaction system that can engage in and follow conversations, albeit within a limited domain. A second is to fully implement an emotional subsystem that can affect the whole range of robot behaviors. A third is to more closely link visual recognition of features--such as shirt color--with the interactions in a natural manner. We came close to that goal this year, but to be smooth it must be integrated with a more generic speech interaction system.

On the navigation front, coverage of the serving area has only been achieved by Mario, mostly because he never stopped to talk. Combining Mario's ability to move in a crowd with a more effective Santino will be difficult, because the robot has to take the initiative and move on.

Finally, the multi-robot system proved to be both entertaining and successful at solving the task. Future competitions should encourage multiple robot interaction--two teams attempted it this year. They will have to deal with the fact that it is difficult for the robots to get to one another, but it should be possible.

In the USR task, the challenge is clear. The autonomous entries covered only a small amount of the test area, mostly because of limitations in their ability to sense and interpret the realities of the situation. The tele-operated entry, on the other, did not give much responsibility to the robots. Building meaningful maps, correctly flagging features or injured people, and simply getting out of the test area within the time limit should be minimal goals for future entries. We believe the techniques exist to accomplish these goals, but their integration in a single package has yet to be done.

References

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

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

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

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

[5] 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. Submitted to *J. Autonomous Robots*.

[6] 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.

[7] 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.

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

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

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

[11] 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.